Lars Erik Tiset Skraastad

Dino

Plattform for forenklet FPGA-design

Masteroppgave i Elektronisk Systemdesign og Innovasjon
Veileder: Per Gunnar Kjeldsberg
Juni 2019

Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske systemer

NTNU
Norwegian University of
Science and Technology

Lars Erik Tiset Skraastad

Dino

Plattform for forenklet FPGA-design

Masteroppgave i Elektronisk Systemdesign og Innovasjon
Veileder: Per Gunnar Kjeldsberg
Juni 2019

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske systemer

Oppgave

Kandidatnavn: Lars Erik Skraastad
Oppgavetittel: Dino – Plattform for forenklet FPGA-design
Medveileder: Joar Rusten, Trådløse Trondheim
Veileder: Per Gunnar Kjeldsberg
Oppgavetekst:
Oppgaven er en fortsettelse av et prosjekt høsten 2018. Målet er å utvikle en plattform som muliggjør enkel design av digitale kretser på FPGA, på samme måte som Arduino-plattformen har gjort det for mikrokontrollere. Plattformen skal være lett å bruke for personer med liten eller ingen erfaring innen digitaldesign, og på en pedagogisk måte gi innføring i digitaldesign. Basert på funn fra prosjektoppgaven og videre evalueringer og eksperimenter skal studenten gjøre vurderinger relatert til hensiktsmessige
- designrepresentasjoner (abstraksjonsnivå og språk/grafikk)
- designverktøy (kretsbeskrivelse og syntese)
- FPGA-plattformer
- eksempeldesign (tilpasset utrent designer som vil starte med digitaldesign)
Deretter skal det utvikles en prototyp som på en god måte demonstrerer egenskapene til plattformen. Denne skal så evalueres med tanke på pedagogisk egnethet.

Sammendrag

Det kan være en stor utfordring å designe digital maskinvare til programmerbare logiske kretser(FPGA). Jo mindre erfaring man innehar, desto mer utfordrende er det. For å kunne utvikle på en FPGA er man avhengig av et utviklingsverktøy som lar deg designe digital maskinvare. De fleste av de eksisterende verktøy i dag er rettet mot industrien og for erfarne utviklere. I denne oppgaven har det blitt undersøkt hvordan brukergrensesnitt til et program kan hjelpe en ufaglært person med å forstå samt utvikle FPGA design. Dette har blitt gjort ved å utvikle to prototyper og disse er evaluert gjennom intervjurunder. Utviklingen av hver prototype er en 4 fases prosess, hvor man tar for seg planlegging, spesifikasjoner, realisering og evaluering av prototypen. Den første prototypen er en visuell fremvisning av utviklingsverktøyet. Ved hjelp av denne prototypen ble designet til verktøyet verifisert. Etter at designet ble verifisert, var neste steg å realisere verktøyet gjennom en kodet prototype. Denne prototypen inneholdt all nødvendig funksjonalitet som trengs for å kunne utvikle på FPGA, men var pakket inn i et minimalistisk og brukervennlig grensesnitt. Gjennom evalueringsrundene som ble gjennomført, var intervjuobjektene tydelige på at verktøyet lyktes med sitt enkle brukergrensesnitt og at verktøyet vil være godt egnet for ufaglærte.

Designing digital hardware for programmable logic circuits (FPGA) can be a big challenge. The less experience you have, the more challenging it is. In order to develop on an FPGA, one is dependent on a development tool that lets you design digital hardware. Most of the existing tools today are aimed at the industry and for experienced developers. This thesis, has been examined how the user interface of a program can help an uneducated person understand and develop FPGA designs. This has been done by developing two prototypes and these are evaluated through interviews. The development of each prototype is a 4-phase process, which deals with planning, specifications, design, and evaluation of the prototype. The first prototype is a visual model of the development tool. By using this prototype, the design of the tool did get verified. After the design was verified, the next step was to realize the tool through a coded prototype. This prototype contained all the necessary functionality needed to develop on the FPGA, and it had a minimalist and user-friendly interface. Through the evaluation rounds that were conducted, the interviewees were clear that the tool succeeded with its simple user interface and that the tool would be well suited for uneducated users.

Forord

Denne oppgaven er basert på resultater fra en prosjektoppgave som jeg gjennomførte høsten 2018[1]. Noe av bakgrunnsteorien som presenteres i denne oppgaven er inspirert fra prosjektoppgaven. Oppgaven er gjennomført for Trådløse Trondheim og jeg ønsker å takke de for all hjelp, spesielt Joar som har bidratt med sin kunnskap når jeg har hatt problemer under utviklingen. Videre ønsker jeg å takke min veileder Per Gunnar for hjelp gjennom denne oppgaven. Per Gunnar har kommet med mange gode forslag når det kommer til oppgavens innhold og struktur.

Denne masteroppgaven har vært veldig interessent å jobbe med, og jeg har lært mye om programvareutvikling og alt det som kreves for å utvikle gode utviklingsverktøy for digital design.

Innhold

Oppgave **v**

Sammendrag **vii**

Forord **ix**

1 Introduksjon **1**
1.1 Prosjektoppgave Dino . . . 1
1.2 Mål og Fremgangsmåte . . . 2
1.3 Viktigste bidrag . . . 2
1.4 Struktur . . . 3

2 Bakgrunn **5**
2.1 FPGA . . . 5
2.2 Digital maskinvareutvikling . . . 7
2.2.1 Abstraksjonsnivå . . . 7
2.2.2 Syntetisering . . . 10
2.2.3 Simulering . . . 10
2.2.4 Plassere og rute . . . 11
2.3 Programvareutvikling . . . 11
2.3.1 Fase 1. Plan . . . 12
2.3.2 Fase 2. Spesifikasjoner . . . 13
2.3.3 Fase 3. Designe . . . 16
2.3.4 Fase 4. Resultat . . . 18
2.4 Nettapplikasjoner . . . 20
2.4.1 Språk for nettapplikasjoner . . . 20
2.4.2 Klient og Tjener . . . 20
2.4.3 HTTP . . . 21
2.4.4 Websocket . . . 21
2.5 Arduino . . . 21

2.5.1 Maskinvare . . . 21
2.5.2 Programvare . . . 22

3 Tidligere Arbeid **25**
3.1 VisualHDL . . . 25
3.2 Concept Engineering . . . 26
3.3 Prosjekt IceStorm . . . 26
3.3.1 Yosys . . . 27
3.3.2 Arachne-PNR . . . 29
3.3.3 IceStorm Toolkit . . . 29
3.4 Netlistsvg . . . 30
3.5 Icarus Verilog . . . 30
3.6 VTR (Verilog to Routing) . . . 30
3.7 IDE8 . . . 31
3.8 Dino Prosjektoppgave . . . 32

4 Prototypeutvikling og Programkrav **35**
4.1 Prototypeutvikling . . . 35
4.2 Programkrav . . . 36
4.3 Intervjuobjektene . . . 39

5 Første prototype: Modell **41**
5.1 Plan . . . 41
5.2 Spesifikasjoner . . . 43
5.3 Designe . . . 45
5.4 Evaluere . . . 46
5.5 Resultat . . . 47

6 Andre prototype: Kodet **49**
6.1 Plan . . . 49
6.2 Spesifikasjoner . . . 51
6.3 Design . . . 53
6.3.1 Første utvikling: Brukergrensesnitt . . . 54
6.3.2 Andre utvikling: Behandle Verilog-kode og generere SVG-fil 56
6.3.3 Tredje utvikling: Generell backend . . . 57
6.3.4 Fjerde utvikling: Ferdigstille og forbedre . . . 59
6.4 Evaluering . . . 66
6.5 Resultat . . . 73

7 Diskusjon **75**
7.1 Frontend . . . 75
7.1.1 Brukergrensesnitt . . . 75
7.1.2 Fremvisning av krets . . . 76
7.2 Backend . . . 76
7.2.1 Verktøy for FPGA utvikling . . . 76
7.2.2 Simulering . . . 77
7.2.3 Utviklingskort . . . 77
7.3 Eksempelkode . . . 78
7.4 Utviklingsprosess . . . 78

8 Konklusjon og videre arbeid **79**

Bibliografi **81**

A Filstrukturen til backend **87**

B Filstrukturen til frontend **89**

Figurer

2.1 Ulike deler av en FPGA(Kilde: E. Monmasson og M. N. Cirste [5]) . 6
2.2 Grunnleggende logiske element(Kilde: F. Piltan et al[8] 6
2.3 Abstraksjonsnivå 7
2.4 Arduinokort(Kilde: Arduino[30]) 22
2.5 Nettbasert IDE(Kilde: Arduino[36]) 23

3.1 Visualisering av design og bølgeform i RTLvision Pro(Kilde: Concept Engineering[42]) 26
3.2 Prosjekt IceStorm design flyt 27
3.3 Yosys data- og kontrollflyt 27
3.4 Arkitekturen til IDE8(Kilde: K. Aalde[57] 32

5.1 Papirmodell 46

6.1 Programflyt 50
6.2 Sammenligning av papirmodell og første kodet prototype 55
6.3 Yosys og NetlistSvg programflyt 56
6.4 Et eksempel på en for stor krets 60
6.5 Programflyt for programmering av FPGA 65
6.6 Startskjerm 66
6.7 Åpne eksempeldesign 67
6.8 Lage en ny fil 68
6.9 Åpne eget design 69
6.10 Verifisere og fremvisning av krets 70
6.11 Demonstrasjon av syntaksfeil 71
6.12 Demonstrasjon: Laste opp kode til FPGA 72

Tabeller

2.1 Korrelasjon mellom prototypeegenskaper og prototypemetode . . . 15
2.2 Verktøy som passer til ulike prototypemetoder 16
2.3 Oversikt over designretningslinjer 17
2.4 Ulike evalueringsverktøy 19

4.1 Programkrav 38

5.1 Protoypekrav 42
5.2 Innhold og trofasthet 43
5.3 Egenskaper for prototype 44

6.1 Protoypekrav 50
6.2 Innhold og trofasthet 51
6.3 Egenskaper for prototype 52
6.4 Meldinger fra frontend 58
6.5 Meldinger fra backend 58

Kapittel 1

Introduksjon

Å lære digital maskinvaredesign kan være utfordrende. Selv på et universitet med gode professorer kan dette være vanskelig å lære. For å kunne lykkes i å forstå dette feltet, er det avgjørende at man forstår de grunnleggende konseptene. Dagens utviklingsverktøy for digital maskinvaredesign er rettet mot industrien. Disse verktøyene er både vanskelige å forstå, og lite pedagogiske. De er laget for rask og god designutvikling for erfarne utviklere, og ikke for læring. Som en ufaglært person med teknologiinteresse og nysgjerrighet kan du skape mye. Du kan løse både store og små problemer, men da trenger du et utviklingsverktøy som kan hjelpe deg i gang. Det er altså et behov for et verktøy som oppmuntrer til læring og som gir brukeren den nødvendig hjelp som trengs.

Ser man på andre fagfelt, finner man mange eksempler på løsninger som har lyktes i å være læringsrike og brukervennlige. Blant de som har lyktes, har vi Arduino[2]. Arduino har klart å gjøre mikrokontroller teknologi tilgjengelig og forståelig for den alminnelige person. Det er mange grunner til at de har klart dette, men en av de viktigste grunnene er utviklingsverktøyet. De har lyktes i å lage et minimalistisk brukergrensesnitt, hvor kun de mest nødvendige funksjonalitetene er synlig og tilgjengelig. En slik løsningen finnes ennå ikke for digital maskinvaredesign. Gjennom denne oppgaven skal vi derfor se på muligheten og prøve å skape dette.

1.1 Prosjektoppgave Dino

Denne masteroppgaven er basert på en prosjektoppgave[1] utført høsten 2018. Målet med den oppgaven var å lage et grunnlag for en utviklingsplattform kalt Dino. Prosjektoppgaven sammenlignet ulike måter å utvikle digitale kretser på,

og presenterte styrker og svakheter med hver av måtene. Oppgaven sammenlignet grafisk utvikling, maskinvarebeskrivende språk(HDL) og programvarespråk(c++, python o.l.). Dette ble gjort for å kartlegge og finne den mest lærerike og intuitive måten å utvikle digitale kretser på. Det ble utført dybdeintervju med fire personer med ulik faglig kunnskap. Resultatene fra intervjuene ble at utvikling av digitale kretser bør utføres ved hjelp av maskinvarespråk, og dette er utgangspunktet for denne masteroppgaven.

1.2 Mål og Fremgangsmåte

Målet med denne masteroppgaven blir å jobbe videre med resultatene fra prosjektoppgaven, og utvikle en fungerende prototype av en nettbasert utviklingsplattform for programmerbare logiske kretser(FPGA). Hovedfokuset gjennom utviklingen av denne plattform vil være brukervennlighet og at plattformen kan hjelpe ufaglærte til å lære digital maskinvaredesign. På grunn av dette vil det bli gjennomført to prototyputviklinger hvor begge blir evaluert av eksterne personer for å sikre godt brukergrensesnitt og brukeropplevelse. Når oppgaven er ferdig vil vi sitte igjen med et web-basert utviklingsplattform, som inneholder både en tekstbehandlingsfunksjonalitet, grafisk fremvisning av kode og en kode-feil detektor.

1.3 Viktigste bidrag

I denne oppgaven utvikles det et utviklingsverktøy som er rettet mot design av digitale og innebygde systemer for bruk på programmerbare logiske kretser(FPGA). Denne oppgavens viktigste bidrag er som følger:

- Oppgaven så på eksisterende utviklingsverktøy som Arduino og lyktes med å reflektere det enkle og intuitive brukergrensesnittet.
- Et brukergrensesnitt som er rettet mot ufaglærte personer.
- Utviklingsplattformen har både grafisk fremvisning av kretsdesign, og syntaks og kode sjekk som bidrar til å øke brukerens forståelse av digital kretsdesign.
- Utviklingsplattformen har all funksjonalitet som trengs for enkelt å programmere en FPGA.

- Utviklingsplattformen inneholder en kode-mal og eksempelkode som med to trykk kan lastes opp på en FPGA.

1.4 Struktur

Gjennom denne oppgaven vil begrepet digital maskinvaredesign bli byttet ut med digital design, siden dette vil øke lesbarehten av oppgaven. Denne rapporten vil være en beskrivelse av utviklingen av utviklingsverktøyet Dino. Strukturen på rapporten vil være organisert slik at leseren vil få en grundig og god forståelse av utviklingen og valgene som blir gjort.

Kapittel 2 tar for seg den nødvendige bakgrunnskunnskap som trengs for å forstå hva som kreves av et utviklingsverktøy, hvorfor de ulike valgene har blitt gjort og hvordan man utvikler programvare ved hjelp av prototyper.

Kapittel 3 presenterer tidligere arbeid som har blitt gjort på dette feltet. Det tar for seg eksisterende løsninger som kan benyttes for å utvikle Dino. I tillegg blir prosjektoppgaven som denne masteroppgaven bygger på presentert.

Kapittel 4 introduserer prototyputviklingen som skal utføres i denne oppgaven og presenterer programkravene for Dino.

Kapittel 5 presenterer utviklingen av første prototype. Prototypen som blir utviklet er en papirmodell, og brukes for å evaluere brukergrensesnittet.

Kapittel 6 tar for seg den siste prototypen. Denne prototype er en fungerende interaktiv kodet prototype, som gjør det mulig å evaluere potensialet til Dino.

Kapittel 7 diskuterer resultatene og valgene som har blitt gjort underveis i utviklingen av Dino.

Kapittel 8 gir en konklusjon på det gjennomførte arbeidet og presenterer videre arbeid.

Kapittel 2

Bakgrunn

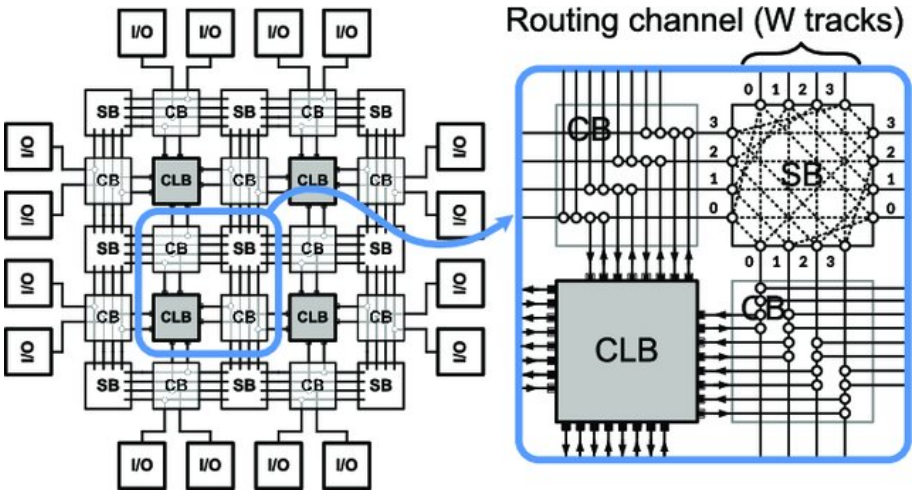
Dette kapittelet tar for seg prinsipper og teori som har blitt benyttet i denne oppgaven. Når man skal utvikle et utviklingsverktøy for digital design, er det viktig å ha kunnskap på flere ulike felt. Først må man ha kjennskap om hvordan man utvikler digital maskinvare. Du trenger kunnskap om de nødvendige stegene som skal til for at du kan gjøre om en idé til et fungerende design. For å få til dette må du være kjent med de ulike måtene å beskrive et design på. Derfor vil vi først i dette kapittelet ta for oss de ulike abstraksjonsnivåene for maskinvare design. Videre vil vi ta for oss de nødvendige verktøytypene som benyttes for at din beskrivelse av et design skal kunne verifiseres og optimaliseres slik at du kan implementere designet ditt på ønsket maskinvare.

For å lage et godt utviklingsverktøy trenger vi også kunnskap og teori om hvordan man utvikler programvare. Det er viktig å jobbe på en effektiv og god måte, som er underbygget av kjente og fungerende modeller. Derfor vil vi senere i dette kapittelet ta for oss den nødvendige teorien om programvareutvikling. Dino vil være en nettbasert utviklingsplattform og det er derfor naturlig å ha litt bakgrunnskap på dette feltet også. Til slutt vil en beskrivelse av utviklingsplattform Arduino bli pressentert.

2.1 FPGA

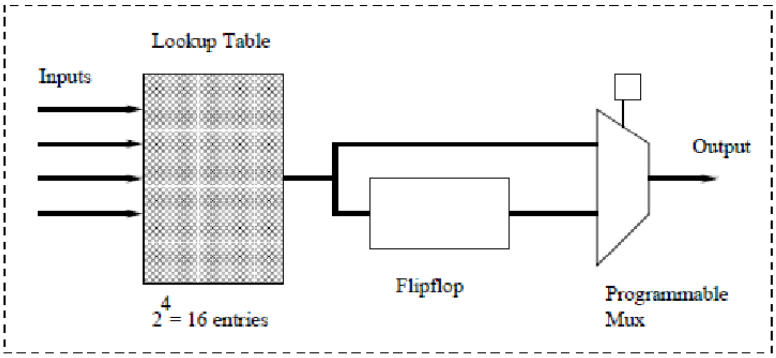
FPGA står for Field-Programmable Gate Array og er en maskinvareteknologi som gjør det mulig å implementere digitale design til programmerbare logiske kretser [3]. FPGA skiller seg fra andre typer brikker ved at man programmerer de fysiske elementene inni. Dette er mulig på grunn av de mange tusen konfigurerbare logiske blokkene som en FPGAer inneholder[4]. Disse blokkene blir ofte kalt CLBer, og ved hjelp av et stort nettverk av programmerbare sammenkoblinger, kan

disse blokkene bli koblet sammen. Avhengig av hvordan disse blokkene med logikk blir koblet sammen så vil de kunne utføre en rekke ulike oppgaver[3]. De ulike delene av et FPGA vises i figur 2.1.



FIGUR 2.1: Ulike deler av en FPGA(Kilde: E. Monmasson og M. N. Cirste [5])

En av de essensielle delene i en FPGA er de konfigurerbare logiske blokkene (CLBer) [6]. CLBene er spredt over hele brikken i en matrise formasjon med et visst antall sammenkoblinger mellom hverandre. En CLB består vanligvis av flere logiske elementer. Disse logiske elementene er igjen bygget opp av to grunnleggende komponenter; LUTs og flip-floper[7], som vist i figur 2.2.



FIGUR 2.2: Grunnleggende logiske element(Kilde: F. Piltan et al[8]

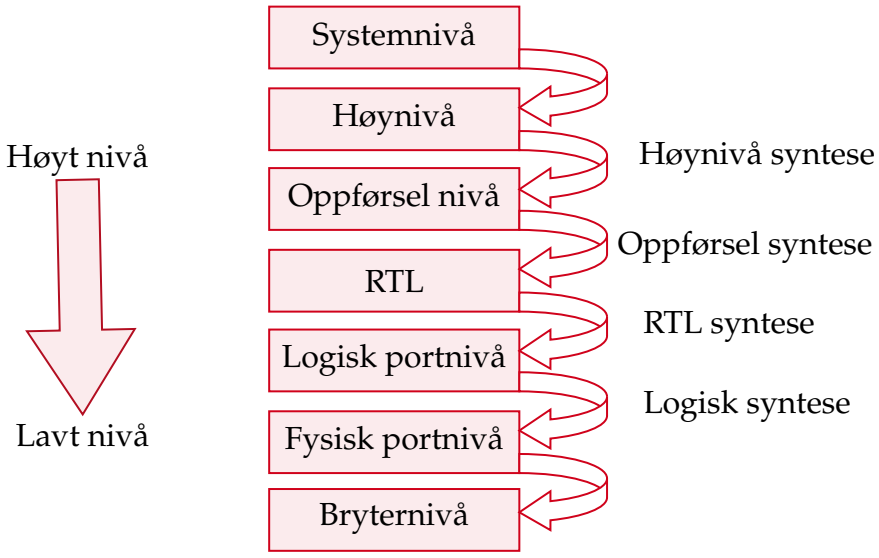
Figur 2.2 viser hvordan et grunnleggende logisk element kan bli pakket, men det er flere måter dette kan bli gjort på. Det som skiller de ulike FPGAene er nøyaktig hvordan disse elementene er pakket[6].

I tillegg til disse CLBer inneholder FPGA også mange inn/ut data-blokker (I/O), som vist i figur 2.1. Disse blokkene gjør det mulig for brikken å kommunisere med omverdenen. Blokkene kan variere fra ADC, DAC, til brytere eller lysdioder. Disse I/O-blokkene kan videre kobles til det store nettverket av sammenkoblinger, og vi vil kunne skape design som kommuniserer med omverdenen.

2.2 Digital maskinvareutvikling

2.2.1 Abstraksjonsnivå

Når man jobber med digital kretsdesign benytter man seg av ulike abstraksjonsnivåer for å kunne beskrive designet på en hensiktsmessig måte. Tidlig i en utviklingsprosess benytter man ofte et høyt abstraksjonsnivå for å beskrive den generelle oppførselen til designet. Videre vil man ha behov for å beskrive designet på en mer nøyaktig og presis måte, da blir man nødt til å benytte seg av et lavere abstraksjonsnivå. Når overgangen fra et høyt nivå til et lavere nivå utføres av en automatisk programvare, kalles denne prosessen å syntetisere[9], [10].



FIGUR 2.3: Abstraksjonsnivå

Dette betyr at et synteseverktøy vil konvertere en høynivå representasjon av en krets til en lik funksjonell krets, men som er presentert på et lavere abstraksjonsnivå. En synteseprosess kan utføres på mange ulike nivåer. De ulike abstraksjonsnivåene og de respektive syntesesprosessene er illustrert i figur 2.3. Videre i dette delkapittelet blir de ulike abstraksjonsnivåer presentert. Det er viktig å merke seg at det finnes en rekke ulike definisjoner på abstraksjonsnivåer, og det som presenteres her er en variant. Abstraksjonsnivåene som presenteres her er gjort i henhold til et synteseverktøy kalt Yosys[9], da dette er verktøyet som blir benyttet i denne oppgaven.

Systemnivå

Systemnivå er det høyeste abstraksjonsnivået og tar for seg det store bilde i et system. Man ser på hva som skal designes og hva som trengs av komponenter for å realisere det[11]. På dette nivået benytter man ofte tradisjonelle språk som C og C++ eller Matlab, og det utføres sjeldent syntese på dette nivået[9].

Høynivå

Høynivå er neste abstraksjonsnivået og man lager en litt mer detaljert beskrivelse av systemet. Dette nivået blir også referert til som algoritmenivået[9]. I motsetning til forrige nivå(systemnivå) beskriver man hvordan de ulike komponentene skal oppføre seg ved hjelp av algoritmer, og hvordan disse skal kommunisere. På dette nivået benytter man også tradisjonelle språk som C og C++. I tillegg til disse språkene finnes det også avarter av språkene som er mer rettet mot digital kretsdesign, her finner vi blant annet SystemC. SystemC er et klassebibliotek for C++, som gjør det mulig å kunne modellere og beskrive maskinvare med et programvarespråk som C++[12].

På dette abstraksjonsnivået finnes det syntetiseringsverktøy[13]–[15]. Disse verktøyene syntetiserer høynivå til oppførsel nivå i form av maskinvarebeskrivende(HDL) kode. Dette betyr at utvikleren ikke trenger å kunne HDL selv, siden dette blir overlatt til synteseverktøyet.

Oppførsel nivå

Oppførsel nivå er det nivået hvor man gir systemet avansert data og kontroll flyt. Dette gjørs med dedikerte språk som er rettet mot maskinvaredesign, som

Verilog[16] og VHDL[17]. Når det kommer til modellering på dette nivået er det avgjørende at språket som benyttes innehar egenskapene som kreves for å beskrive data baner og registrere. For Verilog kalles dette alltid-blokker(always-blocks), og for VHDL heter dette prosessblokker[9]. På oppførsel nivå benytter man noe som heter en sensitivitet-liste for å kontrollere disse blokkene. En sensitivitet-liste er en liste av signaler og forhold som benyttes for å trigge fremgangen i systemet. I Verilog vil det se ut som i kode 2.1

KODE 2.1: Alltid-blokk med sensitivitet-liste

```
always @ ( sensitivitet-liste)
    begin
        // statements
    end
```

RTL(Register Transfer Nivå, Register-overføringsnivået)

RTL står for register-overføringsnivået, og på dette abstraksjonsnivået benytter man kombinatorisk logikk og registre. Registrene står for synkroniseringen av kretsen og er det eneste elementet i kretsen som har egenskapen å lagre data. Den kombinatoriske logikken er den delen av kretsen som utfører de logiske operasjonene[9][18]. Når koden er beskrevet i RTL åpner dette opp for å kunne grafisk presentere designet med kretselementer som register og kombinatoriske celler.[9]. En slik graf, når den er beskrevet som en liste av celler og koblinger blir kalt en netlist.

På dette nivået benytter man på lik linje som nivået over maskinvarebeskrivende språk som Verilog og VHDL. I Verilog vil et RTL kodeeksempel være som følger:

KODE 2.2: Kombinatorisk logikk og registre

```
assign temp = a + b       // Kombinatorisk logikk
always @ ( posedge CLK)  // register
    y <= temp
```

I register-overføringsnivået finner man mange muligheter når det kommer til analysering[9]. For eksempel deteksjon av minne, deteksjon og optimalisering på endelig tilstandsmaskin(FSM), og identifikasjon av delte ressurser[9].

Logisk portnivå

Neste abstraksjonsnivå er logisk portnivå. På dette nivået representeres designet ved hjelp av en netlist som bare består av et lite antall egne celler, som enkle logiske porter (and, or, xor, ol) og register[9].

Fysisk portnivå

Fysisk portnivå bruker bare de fysisk tilgjengelige portene som eksister til målarkitekturen. Avhengig av arkitetkutren, kan dette bety enkle nand-, nor-, not- porter og D register, til mere komplekse celler som komplette halv-adderer[9]. Når man jobber med FPGA design vil den fysiske portnivårepresentasjonen være en netlist med LUTs med egenbestemt utgangsregistre[9].

Transistor

På transistornivå, vil kretsen være presentert i en netlist som benytter enkle transistorer som celler[9].

2.2.2 Syntetisering

Syntese er innen digital maskinvare en prosess utført av et program hvor en beskrivelse av et design blir overført fra et høyere abstraksjonsnivå av designet til et lavere[19]. Ved at et verktøy gjør denne konverteringen, reduseres utviklingstiden, og eliminerer menneskelige feil. Det finnes flere ulike synteseverktøy, og de fleste faller innunder to ulike syntesekategorier. Man har logisk syntese og høynivå syntese. Logisk syntese konverterer maskinvarebeskrivende språk (HDL) til en skjematisk krets[19]. Høy-nivå syntese, konverterer funksjonell eller mikroarkitektrorisk til en beskrivelse i register-overføringsnivået(RTL) [19].

2.2.3 Simulering

Etter at et kretsdesign er beskrevet ved hjelp av skjematisk verktøy eller maskinvarebeskrivende språk(HDL), så trenger man å verifisere at designet oppfører seg som forventet[20]. Å verifisere et design er vanskelig, men en måtte dette kan gjøres på er gjennom design simulering. Når man utfører en slik simulering verifiserer man designet for kun noen spesifikke inn-verdier ved å generere noen

ut-verdier fra en strukturell beskrivelse og sammenligne disse med et forventet ut-verdier som kommer fra en oppførsels beskrivelse[20].

2.2.4 Plassere og rute

Plassere og rute er et steg i utviklingen av maskinvareprogrammering. Som navnet beskriver er dette en prosess som består av to ulike steg, plassere og rute. Det første steget går ut på å bestemme hvor ulike elektroniske komponenter, moduler og logikk elementer skal plasseres[21]. Valget som blir gjort under plasseringen avhenger av tilgjengelig plass. Etter at plasseringen er gjennomført, utføres rute steget. Dette steget går ut på å bestemme nøyaktig design på ledningene som skal koble de plasserte komponentene sammen[21].

2.3 Programvareutvikling

Å utvikle en programvareapplikasjon krever god planlegging, men for å kunne lage en god plan trenger man kunnskap om programvareutvikling. Vi skal derfor i dette kapittelet ta for oss den nødvendige teorien som trengs for å utvikle et program på en effektiv og god måte. Dette kapittelet er basert på konsepter og ideer som er beskrevet i boken "Effective Prototyping For Software Makers" skrevet av J Arnowitz, M Arent og N Berger [22].

Å lage en prototype kan være en avgjørende faktor for at programmet ditt blir vellykket. Det er mange grunner til at en prototype vil hjelpe deg. Ved å bruke tid på denne prosessen vil du kunne få svar på mange viktige spørsmål. Spørsmål som; vil designet virke, vil brukeren like programmet, vil du med tenkt design oppnå det du ønsket. Et av de viktigste spørsmålene er kanskje om designet vil fungere som ønsket? For å få svar på det er det tre ulike faktorer som spiller inn, er programmet lett å bruke, brukerens oppfatning av programmets verdi, og programmets utseende og opplevelse.
For å få svar på disse spørsmålene må vi produsere en prototype. Før man starter på dette vil det være smart å starte med å lage en liste over hva man mener skal være en del av det endelige produktet, dette blir kalt programvarekrav. For å ha en effektiv prototype prosess er det fire faser og 11 steg man bør følge. De fire fasene er: Plan, Spesifikasjon, Design og Resultat. For hver av disse fasene vil det være mellom 2-3 steg, dette vil vi beskrive grundigere senere i kapittelet. Når har

laget nok protyper, og man er fornøyd med resultatet kan man begynne å levere det endelige programmet.

2.3.1 Fase 1. Plan

Dette er den første fasen, og brukes for å kartlegge og bestemme kravene til prototypen. Du skal i denne fasen bestemme hvilken deler av det ønskede programmet som bør og ikke bør være en del av prototypen. Fase 1. kan bli delt opp i tre steg; verifisere krav, lage program flyt og spesifisere innhold og trofasthet.

Steg 1. Verifisere krav

Dette steget går ut på å bestemme seg for prototypekravene. Når man bestemmer seg for prototypekravene trenger ikke disse være det samme som programvarekravene. Det er viktig at man trekker ut de kravene som gjør at prototypen sjekker de viktigste egenskapene ved det ønskede programmet. Når du velger deg prototypekrav bestemmer du deg for hva som vil være hovedfokuset med prototypen.

Steg 2. Lage programflyt

For at det skal være lett å lage en prototype, er det viktig å lage en programflyt. Denne programflyten skal beskrive hvordan brukeren navigerer seg rundt i programmet og hva som skjer når brukeren trykker ulike steder i programmet. Programflyten kan være smart å bruke slik at man ser hvor liten eller stor del av systemet som blir aktivert når bestemte knapper blir trykket på.

Steg 3. Spesifisere innhold og trofasthet

Når man lager en prototype er det vanlig å spesifisere trofastheten til innholdet i prototypen. Ulike egenskaper ved prototypen kan enten ha høy eller lav troverdighet. Hvis en egenskap ved prototypen har høy troverdighet betyr dette at denne egenskapen er veldig lik på prototypen som den vil være i det endelige programmet. Siden en protype skal være en etterligning og test av det endelige programmet, er det naturlig å fokusere på spesifikke deler av programmet, og at prototypen demonstrer den ønskelige funksjonen og ikke noe mer.

2.3.2 Fase 2. Spesifikasjoner

I denne fasen skal vi ta for oss resultatene som ble utledet i forrige fase. Fasen starter med at vi må definere prototype karakteristikk og deretter skal vi velge oss et prototypeverktøy. Denne fasen består av tre steg.

Steg 4. Bestemme seg for riktig prototypegenskaper

Fjerde steg er å bestemme seg for riktig prototypegenskaper, og disse egenskapene er viktig å velge før en prototypemetode blir bestemt. Ved å bestemme prototypegenskapene tidlig vil det være enklere å bestemme seg for hvilken type prototype som trengs og hva slags egenskaper prototypen bør inneha.

Det er totalt 8 ulike prototypegenskaper vi kommer til å vurdere. Disse er presentert under, sammen med en forklaring på hver av de:

1. **Målgruppe:** Det er viktig å være klar over målgruppen, og at utvikleren og sluttbrukere ikke nødvendigvis faller under samme gruppen. Av den grunn må man være bevisst når evaluering av prototype skal utføres, slik at sluttbrukerens mening blir vurdert.

2. **Stadium:** Man må også være bevisst på hvilket stadium av programmet som prototypen skal reflektere. Her skiller man mellom **tidlig**, **midt** og **sen**. Ved et tidlig stadium fokuserer man ofte på det store bildet, men i et sent stadium er prototypen ganske lik sluttproduktet.

3. **Hastighet:** Hvor raskt og grundig trenger utviklingen av prototypen være. Her skiller man mellom **rask** og **flittig**.

4. **Levetid:** Hvor lenge skal prototypen bli brukt. Er dette en prototype som raskt blir forkastet eller skal den brukes i en lang periode. For denne egenskapen skiller man på **kort**, **middels** og **lang**.

5. **Uttrykk:** Hvor abstrakt eller konkret skal prototypen være. Ønsker man å vise funksjonalitet eller noe visuelt eller kanskje en kombinasjon. De to ulike typer uttrykk man kan velge mellom er **konseptuell** eller **eksperimentelle**.

6. **Stil:** Stilen på prototypen handler om hvor mye brukeren kan interagere med prototypen. Her skiller man på om prototypen er **fortelling** eller **interaktiv**.

7. **Medium:** Når du skal lage en prototype kan man velge hvordan denne skal fremvises, skal den være **fysisk** eller **digital**.

8. **Trofasthet:** Hvor troverdig og lik det endelige programmet vil prototypen være. Den kan ha enten **lav**, **middels** eller **høy** troverdighet.

Steg 5. Valg av metode

Nå som vi har en oversikt over de ulike egenskaper en prototype kan ha, er det på tide å bestemme seg for hvilken prototypemetode som passer. De 8 mest vanlige prototypemetodene er presentert under:

1. **Kortsortering:** En abstrakt metode som brukes for å få oversikt over struktur og informasjon om programmet.

2. **Rammeverk:** En slik prototype viser høynivå illustrasjon av struktur og interaksjon av modellen.

3. **Storyboard:** Brukes tidlig i en utviklingsprosess og går ut på å sette seg inn i hvordan brukeren vil bruke programmet, og på den måten få en oversikt over hva som trengs .

4. **Papir:** Dette er en interaktiv modell av programmet som forteller om hvordan brukergrensesnittet er og hvordan programmet oppfører seg ved brukerinteraksjon.

5. **Digital:** Detter er en tilsvarende prototype som Papir, men her benyttes et digitalt verktøy som gjør at brukerinteraksjon gir en fysisk endring i modellen.

6. **Video:** Dette er en visuell representasjon av programmet som viser hvordan det er tenkt å fungere.

7. **Wizard-of-Oz:** Dette er en interaktiv modell hvor funksjonaliteten til programmet ikke eksisterer, men hvor disse heller blir etterlignet.

8. **Kode (inkludert skripting og HTML):** Denne metoden går ut på å lage prototypen ved hjelp av programmering og script-språk.

Når du skal bestemme deg for hvilken av disse metodene man skal benytte kan det være smart å lage en tabell med alle prototypegenskapene som vi så på

tidligere, og markere viktigheten på hver av disse egenskapene. Gjør man det kan man ende med en tabell som vist i Tabell 2.1. Her ser vi en oversikt over alle metoder og korrelasjonen mellom disse og de ulike prototypeegenskaper.

TABELL 2.1: Korrelasjon mellom prototypeegenskaper og prototypemetode

Egenskaper		**Metoder**							
		Kort	Ramme-verk	Story board	Papir	Digital	Video	Wizard of Oz	Kode
Mål-gruppe	Intern	x	x	x	x	x	x		x
	Ekstern	x		x	x	x	x	x	x
Stadie	Tidlig	x	x	x			x		
	Midt		x	x	x	x	x	x	
	Sen					x			x
Hastighet	Rask	x	x	x	x	x	x	x	
	Flittig					x	x		x
Levetid	Kort	x	x						
	Middels		x	x	x	x	x	x	
	Lang					x	x		x
Uttrykk	Konsept	x	x	x	x	x	x	x	
	Eksper				x	x	x	x	x
Stil	Fortelling		x	x			x		
	Interaktiv	x			x	x	x	x	x
Medium	Fysisk	x	x	x	x				
	Digital	x	x	x		x	x	x	x
Trofasthet	Lav	x	x						
	Middels		x	x	x	x	x	x	
	Høy			x		x	x	x	x

Steg 6. Valg av prototypeverktøy

Etter at man har bestemt seg for hvilken metode som passer for din prototype, trenger du et verktøy. Dette verktøyet skal hjelpe deg med å lage en best mulig prototype. I Tabell 2.2 ser du en oversikt over noen verktøy og hvilken metode disse passer til. I denne tabellen betyr "x" passende, "NA" ikke tilgjengelig og blankt betyr ikke passende.

TABELL 2.2: Verktøy som passer til ulike prototypemetoder

	Metoder							
Verktøy	Kort	Ramme-verk	Story board	Papir	Digital	Video	Wizard of Oz	Kode
Word	NA	x	x	x		NA	NA	NA
PowerPoint	NA	x	x		x	NA	NA	NA
Excel	NA	x		x	x	NA	NA	NA
Papir	x	x	x	x	x	NA	NA	NA
Photoshop	NA	x	x	x	x	NA	NA	NA
FrontPage	NA	x			x	NA		x
Flash	NA			NA	x	x	x	x
Director	NA			NA		X	x	x
VisualStudio	NA		NA	NA	x	NA	x	x

2.3.3 Fase 3. Designe

Nå som man har spesifisert prototypestrategi, vil det være på tide å begynne å lage prototypen. I denne fasen er fokuset å lage et godt design, på en effektiv måte. Dette gjøres i to steg; formulere designkriterier og lage prototypen.

Steg 7. Formulere designkriterier

Dette steget prøver å gi en oversikt over hva som kan være god praksis når det kommer til grensesnitt design, visuell design og brukbarhet. Målet her er å få deg til å bli bevisst på designvalgene du gjør. Tabell 2.3 viser en rekke med ulike retningslinjer. For å lage et godt design bør man lage en prioritering av disse retningslinjene, og velge ut de som passer din prototype best. Retningslinjene er fordelt under brukergrensesnitt og visuell design, og hver retningslinje er kategorisert under organisatorisk eller retningsbestemt retningslinje. Organisasjonsretningslinjene skal hjelpe deg med å organisere brukergrensesnittet ditt, det vil si for eksempel hvordan du kan gruppere relevante elementer. Retningsbestemte retningslinjer skal hjelpe med flyten av sideoppsettet, altså hvor på skjermen informasjon bør plasseres slik at brukeren raskt vil oppfatte informasjonen.

TABELL 2.3: Oversikt over designretningslinjer

Retningslinjer	Kategori	Forklaring
Visuell design		
Informasjonsflyt	Retningsbestemt	Hvor vil øynene til brukeren bevege seg.
Rutenett-basert	Organisatorisk	Gruppering av ulike elementer.
Rytme og mønster	Retningsbestemt	Fleksibelt og logisk mønster
Likhet og variasjon	Organisatorisk	Ensartet design
Typografisk struktur	Organisatorisk	Lesbarheten
Balanse	Retningsbestemt	Balanse mellom symmetri og asymmetri
Logisk gruppering	Organisatorisk	
Brukergrensesnitt		
Progressiv avsløring	Retningsbestemt	Varier mengde vist data
Effektivitet	Retningsbestemt	Enkel navigering
Fitt's lov	Retningsbestemt	Det som brukes mye skal være lett tilgjengelig
Lærbarhet	Retningsbestemt	Prototypen må være enkel å lære seg
Snakk publikums språk	Organisatorisk	Reflekter brukerens vokabular og uttrykksmåte
Vis eksplisitt krevde handlinger og felt	Organisatorisk	Enkelt å forstå hva man må gjøre for at enkelte handlinger skjer
Internasjonal følsomhet	Organisatorisk	Global forståelse av innholdet
Universell tilgjengelighet	Organisatorisk	Enkelt for alle å bruke
Bruker bør føle seg i kontroll	Organisatorisk	Føle man vet hva som vil skje når man utfører handlinger
Minimer kognitiv belastning	Organisatorisk	Skal være enkelt å huske hvordan ulike handlinger fungerer
Tilfredshet	Organisatorisk	Behagelig å bruke

Steg 8. Lage prototypen

Steg 8. går ut på å lage prototypen. Da skal du ved hjelp av alle de foregående stegende være kapabel til å utvikle en kvalitets prototype på en effektiv måte.

2.3.4 Fase 4. Resultat

Til slutt har vi resultatfasen. Det er nå på tide å gjennomgå og validere designet.

Steg 9. Gjennomgå design

Når man skal gjennomgå designet er det flere viktige faktorer som spiller inn. En av faktorene som er viktig å være klar over er forventingene til de som skal evaluere designet. Det er viktig at man informerer de som skal evaluere om hvilket steg i programvareprosessen man er i, og hvor ferdig innholdet egentlig er. Når forventingene er satt, vil neste steg være å bestemme seg for hvordan man skal presentere prototypen. Da kan det være hensiktsmessig å lage seg en agenda som de som skal evaluere designet får, slik at de kan være forberedt på hva som møter dem. Videre er det viktig å være god til å presentere. Dette er en egenskap noen har mer av enn andre, men så lenge man er godt forberedt kommer man langt.

Steg 10. Validere design

Når designet skal valideres, vil det være nødvendig å velge et evaluerings verktøy. En liste over hvilken verktøy som passer til hvilken prototypemetode er beskrevet i Tabell 2.4. Som tidligere nevnt betyr “x” passende, “NA” ikke tilgjengelig og blankt betyr ikke passende. Fra denne tabellen kan man enkelt velge et passende evalueringsverktøy som gjør evaluringen mest mulig hensiktsmessig.

TABELL 2.4: Ulike evalueringsverktøy

	Metoder							
Verktøy	Kort	Ramme-verk	Story board	Papir	Digital	Video	Wizard of Oz	Kode
Intern Intervju	x	x	x		x	x	x	x
Ekstern Intervju	NA		x	NA	x	x	x	x
Brukervennlighet test	NA	NA	NA	x	x	NA	x	x
Fokus gruppe testing	x		x	x	x	x	x	x
Fagfellevurdering	x	x	x	x	x	x	x	x
Bruker tilbakemelding	NA		x	x	x	x	x	x
Undersøkelse	NA	NA	NA	NA	x	NA	NA	x
Kognitiv gjennomgang	NA	x	NA	x	x	NA		x

2.4 Nettapplikasjoner

2.4.1 Språk for nettapplikasjoner

Når man lager en nettapplikasjon benytter man ulike språk for å beskrive ulike deler av en applikasjon. Når man skal beskrive den visuelle delen av en nettapplikasjon benytter man seg ofte av HTML, CSS og JavaScript.

HTML

HTML er et språk som benyttes når man skal presentere hypertekst-dokumenter på internett[23]. HTML er det språket man benytter for å presentere tekst på en nettside.

CSS

CSS er et format som benyttes for å bestemme stilen til et HTML-dokument[24]. Ved hjelp av CSS kan man kontrollere layoutegenskapene til dokumentet. Det betyr å endre skriftstørrelse, linjeavstand og farger.

JavaScript

JavaScript er et høynivå-programmeringsspråk som kan benyttes på klientsiden av en nettapplikasjon[25]. Da benytter man dette språket for å kunne gjøre endringer på innholdet eller oppførselen i nettsiden.

2.4.2 Klient og Tjener

Tjener(også kalt Server) er den delen av et system som leverer nettjenester til en klient[26]. En klient kan anses å være den delen av et system som skal motta data fra en tjeneren, det kan være en datamaskin, mobiltelefon eller et annet program. Når en tjener benyttes i en nettapplikasjon er oppgaven til tjeneren å sende ut tjenester til nettsider, og klienten bruker en nettleser for å kunne hente disse tjenestene.

2.4.3 HTTP

HTTP er en kommunikasjonsprotokoll som benyttes når en klient og tjener skal overføre HTML-dokumenter[27]. Når man benytter seg av en http i en URL-adresse signaliserer dette at klienten ønsker å benytte seg av denne kommunikasjonsprotokollen.

2.4.4 Websocket

Websocket er en protokoll som muliggjør toveiskommunikasjon mellom en klient og tjener[28]. Denne protokollen benyttes når man ønsker en mekanisme for en toveiskommunikasjon mellom en nettapplikasjon og tjener, uten å måtte åpne mange HTTP koblinger.

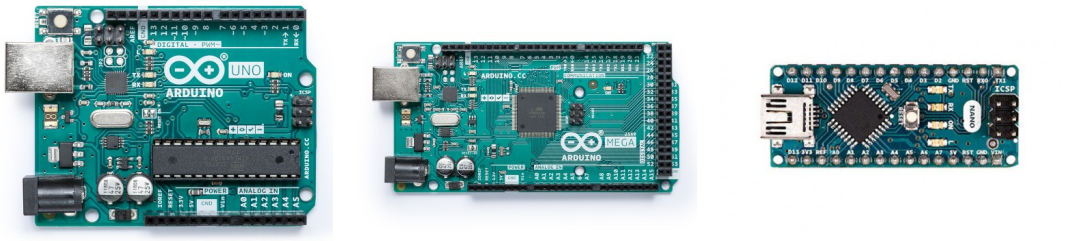
2.5 Arduino

Arduino er en av de største utviklingsplattformene for laging av prototyper av elektronikk. Arduino er et verktøy med fokus på rask utvikling og brukervennlighet for personer med begrenset kunnskap om programmering og elektronikk[2]. Det er flere grunner til at Arduino har lykkes. En av grunnene er at de har klart å vise hvor mye man kan bruke elektronikk og programmering til, og hvor lett det kan være. De har en filosofi som sier at elektronikk kan gjøres enkelt uten å måtte fjerne all kompleksitet [29]. I tillegg har Arduino priset sine mikrokontrollere lavt, noe som gjør det tilgjengelig for alle.

For å kunne utvikle innenfor Arduino-økosystemet, trenger du en mikrokontroller og et utviklingsmiljø (IDE). Arduino tilbyr alt dette. Det du trenger å kjøpe er et Arduino utviklingskort, og laste ned Arduino sitt eget gratis IDE. Sammenlignet med andre utviklingsplattformer er Arduino rettet mot den kreative, nysgjerrige og lekende ikke-ingeniør. Derfor måtte de gjøre det enklest mulig for brukeren, og dette er noe de har klart.

2.5.1 Maskinvare

Arduino sine utviklingskort er kompakte, de består av en mikrokontroller, I/O-porter og enkle kretser som kobler portene til mikrokontroller[31]. Arduino tilbyr mange forskjellige kort, de mest populære er vist i figur 2.4[32]. De har alt fra små



(A) Uno (B) Mega (C) Nano

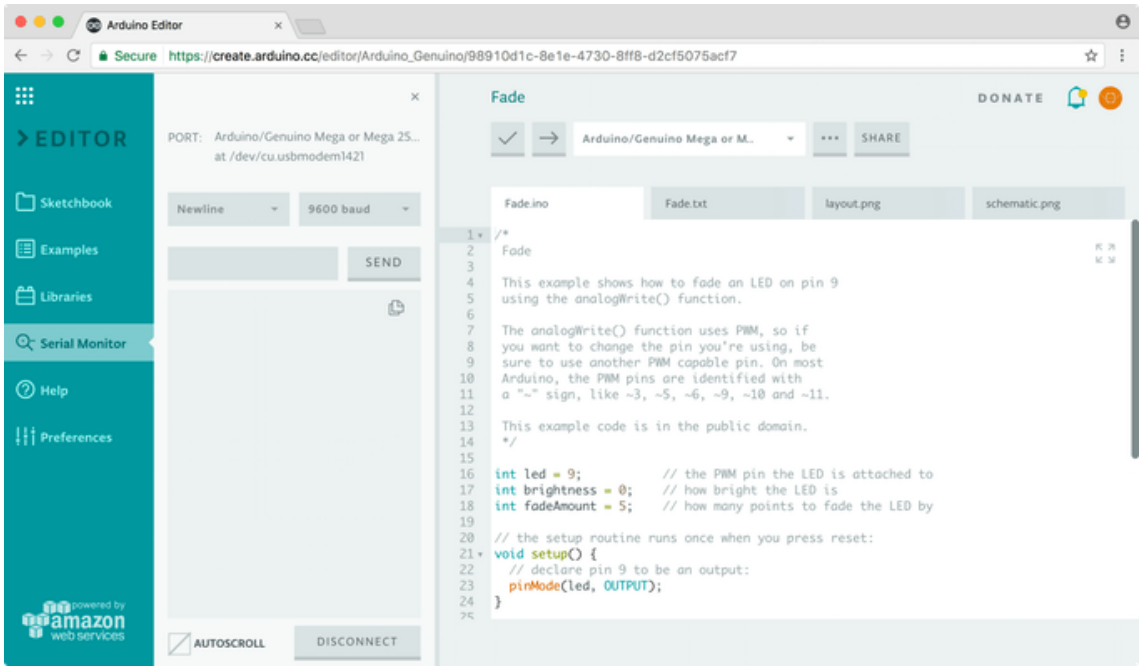
FIGUR 2.4: Arduinokort(Kilde: Arduino[30])

og enkle kort til store og kraftige. Avhengig av hvilket kort man velger finner man mange spennende egenskaper som gjør dem attraktive for ulike prosjekter. Kortene varierer fra hverandre med antall I/O-porter, minne, lyd, pris og størrelse. Hvis prosjektet ditt krever enten, mange I/O-porter, mye minne eller lyd, så vil de mer kraftige kortene som Arduino Mega (figur 2.4b) være passende [33]. Hvis du ikke trenger alle disse funksjonene, kan Arduino Uno (figur 2.4a) eller Nano (figur 2.4c) passe bedre. Av disse er Nano den minste og billigste [30].

2.5.2 Programvare

I tillegg til maskinvare tilbyr Arduino også et eget integrert utviklingsmiljø(IDE). Denne IDEen forsøker å gjøre utviklingen av kode enklest mulig uten å fjerne brukerens fleksibilitet. IDEet er minimalistisk og intuitivt. Det fungerer på Windows, Linux og Mac OS, i tillegg eksisterer det en nettbasert utgave[34], som vist i figur 2.5. Verktøyet er utformet slik at brukeren enkelt kan organisere, redigere, kompilere og laste opp koden til kortet[29]. Verktøyet tilbyr også en seriell skjerm som gjør det mulig å sende og motta data fra kortet[35].

Koden du skriver, kalles en skisse [37], og språket du bruker er Arduino sitt eget programmeringsspråk[38]. Arduino programmeringsspråket er C/C++ lignende språk med et kjernebibliotek som består av AVR C/C++ funksjoner [29]. De innebygde Arduino-funksjonene gjør at du kan utvikle koden din i C/C ++, men de abstraherer vekk lavnivådelen. Uten disse funksjonene må du bruke datarket for mikrokontroller for å oppnå samme funksjonalitet[29]. IDEen kommer også med mange innebygde eksempler, og du har mulighet til å legge til flere biblioteker for å utvide funksjonaliteten ytterligere.



FIGUR 2.5: Nettbasert IDE(Kilde: Arduino[36])

Kapittel 3

Tidligere Arbeid

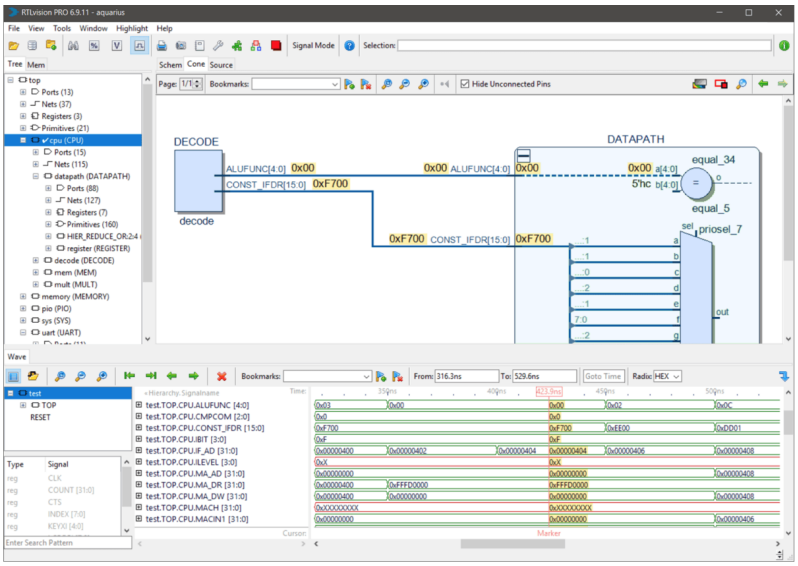
Dette kapittelet tar for seg lignende prosjekter som Dino. I tillegg til dette blir det presentert tidligere arbeid som kan være til nytte når man skal lage et utviklingsverktøy. Å lage et utviklingsverktøy er en omfattende oppgave, som krever mange ulike avanserte delsystemer. Man trenger for det første et brukergrensesnitt som lar brukeren navigere seg rundt i programmet, og som lar brukeren kunne skrive ønsket kode. I tillegg til dette er det behov for all logikk som utføres i bakgrunnen. Det vil være alt fra filhåndtering, behandling og feilsøking til syntetisering av kode, og mye mer. Alt dette vil ikke være mulig å utvikle på den avsatte tiden, og det er derfor nødvendig å kartlegge og se på eksisterende løsninger. Derfor vil vi i dette kapittelet se på hvilke løsninger som eksisterer i dag, slik at vi senere kan velge oss de nødvendige løsningene som vil gjøre at Dino får de ønskelige funksjonalitetene.

3.1 VisualHDL

VisualHDL er et integrert utviklingsmiljø (IDE) med hovedfokus på rask FPGA utvikling[39]. Å utvikle FPGA design i VisualHDL gjørs med man et språk som heter THDL++. THDL++ er et kompakt maskinvarespråk med full parallell VHDL semantikk[40]. Dette gjør at det er mulig å kunne utvikle fult parallelle FPGA design med modularitet og fleksibilitet[39]. Språket er bygget på C++ syntaks og støtter objekt orienterte egenskaper, som gjør koden enkel å forstå. Verktøyet er ikke en C++ til FPGA kompilator, men heller et verktøy for kode komplettering, kode navigering, simulering og design visualisering.

3.2 Concept Engineering

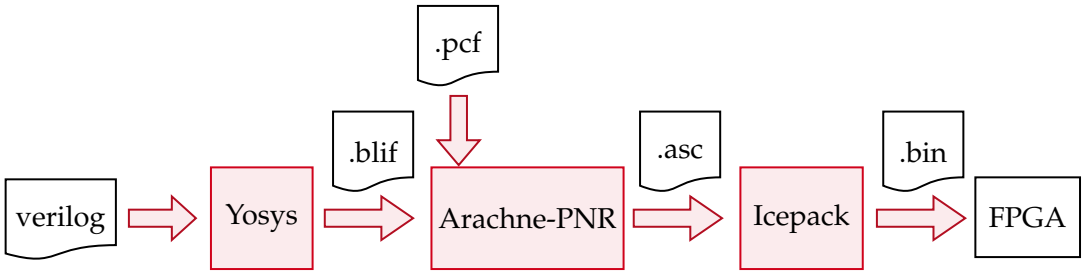
Concept Engineering er et firma som har spesialisert seg på elektronisk system visualisering programvare[41]. De har utviklet flere feilsøkingsprogram hvor de ulike programmene er spesialisert for ulike abstraksjonsnivå av maskinvare design(kapittel 2.2.1). Et av de mest interessante programmene til Concept Engineering er RTLvision Pro, som er en RTL-feilsøker med støtte for både VHDL, Verilog og SystemVerilog. Programmet har en avansert visualisering av kodestruktur og bølgeform, dette er vist i figur 3.1. Programmene har ikke åpen kildekode og er ikke gratis.



FIGUR 3.1: Visualisering av design og bølgeform i RTLvision Pro(Kilde: Concept Engineering[42])

3.3 Prosjekt IceStorm

Prosjekt Icestorm er et åpent kildekodeprosjekt som har som mål å tilby enkle verktøy for å analysere og lage bitstrømfiler for Lattice iCE40 FPGAer. Måten de gjør dette på er ved omvendt konstruksjon(reverse engineering) og dokumentere bitstrøm-formatet til Lattice iCE40 FPGAer[43]. Til nå mener Prosjekt IceStorm at de har klart å omvendt konstruere to FPGA enheter og det er 1K og 8K brikkene til Lattice.

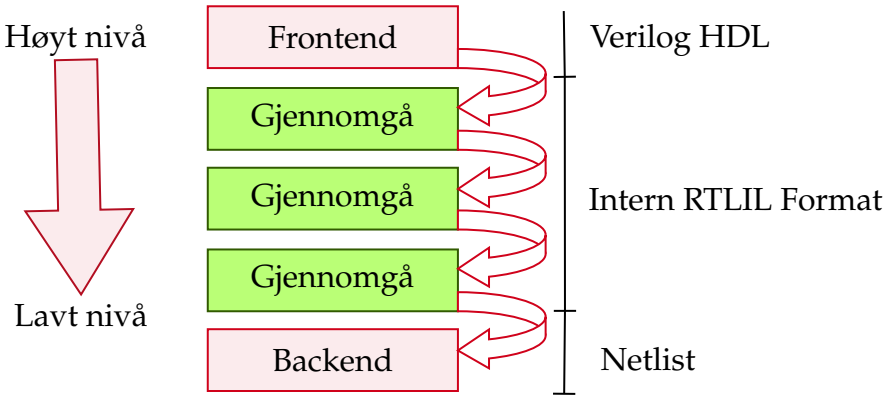


FIGUR 3.2: Prosjekt IceStorm design flyt

Prosjekt IceStorm er bygd opp av tre ulike deler, slik at du som bruker enkelt kan programmere din egen FPGA. Design flyten er presentert i figur 3.2. Den første delen er synteseverktøyet Yosys. Andre del er et plassere-og-rute verktøy som heter Arachne-pnr. Til slutt tilbyr de IceStorm toolkit, som er en samlepakke av mange mindre verktøy som brukes til å programmere FPGAen.

3.3.1 Yosys

Yosys er et syntetiseringsverktøy(kapittel 2.2.2) med åpen kildekode. Bakgrunnen for Yosys, er at de ønsker å tilby et tilgjengelig, universelt og leverandøruavhengig syntese verktøy, hvor brukeren selv kan integrere egne syntesealgoritmer. De mener at et åpent syntetiseringsverktøy vil åpne opp for økt forskning av elektronisk designautomatisering(electronic design automation (EDA))[44].



FIGUR 3.3: Yosys data- og kontrollflyt

Hovedmålet med Yosys er å syntetisere Verilog HDL til netlist[45]. Yosys er et modulært program[9], og data- og kontrollflyten til verktøyet er presentert i figur 3.3. Yosys er bygget opp av flere småsystemer, og det første småsystemet i

Yosys er frontend. Dette er den delen av verktøyet som tar imot brukerdata og sender den videre i programmet. Yosys har til nå støtte for store deler av Verilog-2005 i sitt frontend, men vil i fremtiden også få støtte for VHDL[44]. Videre har Yosys flere småsystemer som har som oppgave å behandle data, disse småsystemene kalles for passes(gjennomgå). Til slutt finner vi backend som er den delen av verktøyet som returnerer ønsket utdataformat.

Når man bruker Yosys er alle de nevnte småsystemene tilgjengelig for brukeren. Disse er tilgjengelig via kommandoer, slik at brukeren selv kan bestemme flyten til synteseverktøyet. En liten liste av tilgjengelige kommandoer er pressentert i Kode 3.1.

KODE 3.1: Yosys kommandoer[45]

```
# read design using verilog frontend
read_verilog mydesign.v

# analyze design hierarchy
hierarchy -check -top mytop

# map always-blocks to RTL netlists
proc; opt

# optimize FSM state encodings
fsm; opt

# map design to the built-in
# logic-level cell library
techmap; opt

# write verilog netlist
write_verilog synth.v
```

Når man bruker Yosys som en del av Prosjekt IceStorm, utfører man alle ønskelige optimaliseringer og syntetiserer deretter Verilog-fil til en .blif fil. BLIF står for Berkeley Logic Interchange Format, og er en filtype som benyttes når man ønsker å beskrive en krets på logisk nivå(Kapitel 2.2.1) i tekst form[46]. Yosys har også en mulighet for grafisk fremvisning. I tillegg kan man konvertere denne

BLIF-filen til et JSON-objekt som kan være til nytte når man skal sende dataen over nett.

Fokuset til Yosys har frem til nå vært funksjoner og behandling av gyldig kode, og ikke feilrapportering ved ugyldig kode[47]. Dette betyr at verktøyet ikke returnerer årsaken, når en syntetisering ikke klarer å gjennomføres.

3.3.2 Arachne-PNR

Arachne-PNR er en del av Prosjekt IceStorm, og er et plassere og rute verktøy(kapittel 2.2.4)[48]. Siden dette verktøyet er utviklet som en del av Prosjekt IceStorm er både inndata og utdata satt til spesifikke formater slik at bruken av verktøyet vil være enklest mulig. For at Arachne-PNR skal kunne plassere og rute, trenger verktøyet informasjon om utviklingskortet. Dette er informasjon om hvor lysdiodene og digitale ut/inn festepunkter er koblet til FPGA brikken. Informasjonen lagres i en pcf-fil. Arachne-PNR tar inne denne pcf-filen sammen med den syntetiserte og optimaliserte kretsen som Yosys produserer og eksporterer til en blif-fil, dette er vist i figur 3.2. Når Arachne-PNR har utført sine algoritmer for plassering og ruting returneres en ASCII representasjon av kretsens bitstrøm. Algoritmene som utføres er rettet mot Lattic ICE40 FPGAene og den produserte bitstrømmen vil derfor kun være kompatibel med denne FPGA familien[48].

3.3.3 IceStorm Toolkit

IceStorm Toolkit er en samlepakke av små programmer som skal hjelpe brukeren med å overføre bitstrømmen som kommer fra Arachne-PNR til ønsket Lattic ICE40 FPGA[43]. De to viktigste programmene i denne pakken er IcePack og IceProg. IcePack er et program som konverterer ASCII filen til en binærfil(.bin) som benyttes når man skal programmere en FPGA. IceProg er det programmet man bruker når man sitter med binærfilen og ønsker å overføre den til FPGAen, slik at din krets vil fysisk kjøre på maskinvaren.

3.4 Netlistsvg

Netlistsvg er et verktøy som konverterer en JSON RTL netlist til SVG skjematikk[49]. Bakgrunnen for Netlistsvg er å lage en enkelt og et lett tilgjengelig verktøy for fremvisning av digitaldesign. Verktøyet er en utvidelse av Yosys sin skjematiske fremvisning. Yosys er som nevnt tidligere et rammeverk som kan konvertere Verilog til RTL netlist. Yosys kan produsere netlisten som et JSON objekt, og Netlistsvg er laget for å kunne tegne SVG skjematikk fra disse JSON objektene. For å generere et slikt SVG bilde bruker man kommandoen som presentert i kode 3.2.

KODE 3.2: Netlis til SVG kommando[49]

```
netlistsvg input_json_file [-o output_svg_file]
```

3.5 Icarus Verilog

Icarus Verilog er et Verilog simulerings- og synteseverktøy[50]. Verktøyet er ikke et endelig produkt, men er allikevel et stabilt og velfungerende. Icarus Verilog er et program med åpen kildekode og er gratis å bruke. Verktøyet er tilgjengelig for en rekke operativsystemer, som Linux, FreeBSD, Windows og Mac OS.

Bakgrunnen for programmet var at det skulle være en simulator, men har i senere tid også fått støtte for syntetisering. For å utføre simuleringer blir Verilog-kode kompilert til en kjørbar fil. Under denne kompileringen utføres det også en kode og syntaks sjekk. Dette kan være til stor nytte når man har feil i koden som man selv ikke ser. For å kompilere en fil kan man kjøre kommandoen som er beskrevet i kode 6.7.

KODE 3.3: Icarus kommando for kompilering av fil[50]

```
iverilog -o hello hello.v
```

3.6 VTR (Verilog to Routing)

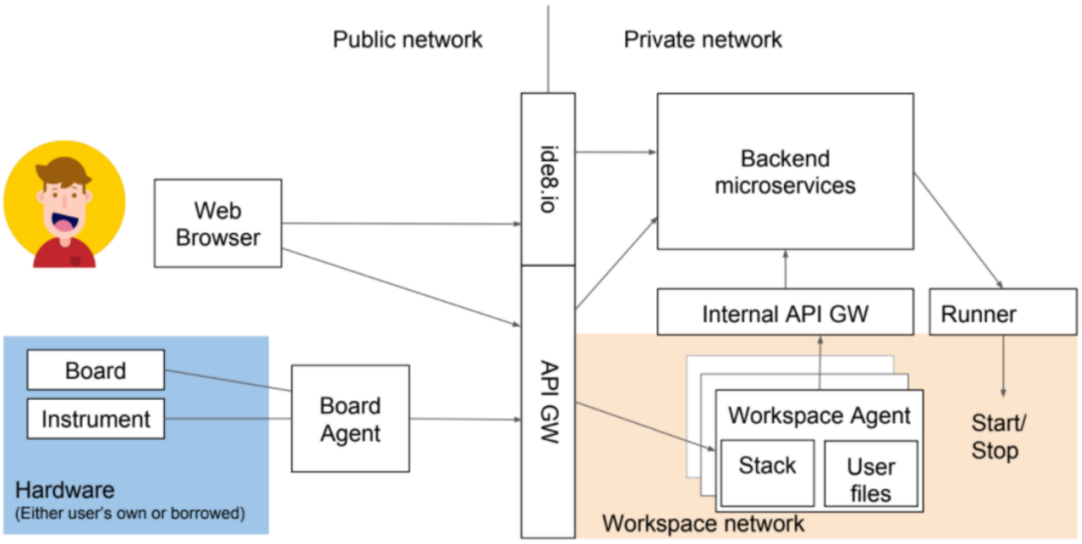
Verilog to Routing(VTR) er et globalt samarbeid hvor målet er å lage et gratis og åpent kildekoderammeverk for innhenting av FPGA arkitektur[51]. Programflyten til VTR er at brukeren skal sende en Verilog fil med beskrivelse av sin digitale

krets, sammen med ønsket FPGA arkitektur. Videre skal VTR utføre syntese gjennom verktøyet Odin II[52], optimalisering og teknologi kartlegging med ABC[53] og til slutt pakke, plassere og rute med et verktøy kalt VPR[54].

3.7 IDE8

IDE8 er et nettbasert utviklingsmiljø for å prototype elektronikk. IDE8 er utviklet av Trådløse Trondheim[55] og er enn så lenge under utvikling og er derfor ikke et ferdig produkt. Bakgrunnen for IDE8 er å gjøre utviklingen av protyper enkelt og lett tilgjengelig for brukeren. Trådløse Trondheim ønsker å tilby et komplett og dynamisk utviklingsmiljø. Normalt må man som utvikler gå til anskaffelse av alt fra mikrokontrollere og sensorsystemer, i tillegg vil man da bli nødt til å benytte spesifikke utviklingsverktøy for å kunne programmere mikrokontrolleren. Verktøyene man ofte bruker er designet for å gjøre spesifikke oppgaver, så hvis man ønsker ekstra funksjonalitet som for eksempel måle strømtrekket, trenger man ekstra verktøy. IDE8 har som mål å løse denne problemstillingen ved at brukeren via nettleseren skal kunne logge seg inn i et utviklingsmiljø hvor man vil få tilgang til en rekke ulike maskinvare konfigurasjoner og utviklingsplattformer. IDE8 kan bli beskrevet som en PaaS(Plattform som en tjeneste)[56] siden det er et skybasert utviklingsmiljø med flere ulike ressurser for å kunne utvikle alt fra enkle til avanserte elektroniske systemer.

Trådløse Trondheim fokuserer på at IDE8 skal være et dynamisk verktøy. Det vil si at brukeren selv bestemmer størrelsen og mengden maskinvare og programmvare-komponenter man ønsker å bruke. Utviklingsmiljøet som brukeren jobber i kalles et Arbeidsområde(Workspace). En bruker kan ha mange arbeidsområder, og et arbeidsområde bygger på ferdige stabler(stacks) som brukeren kan velge mellom. Når man velger en stabel, så velger man hvilket tjenester og verktøy som skal være tilgjengelig for deg når du er i ditt arbeidsområde. Stablene man velger mellom er designet for spesifikke maskinvarer, typer applikasjoner, sensorer eller ulike eventer. Dette vil si at en stabel kan være designet for en IoT applikasjon, et kamera, eller en FPGA. Hvordan IDE8 er organsiert er vist i figur 3.4



FIGUR 3.4: Arkitekturen til IDE8(Kilde: K. Aalde[57]

3.8 Dino Prosjektoppgave

Dino var i høsten 2018 en prosjektoppgave som ble gjennomført ved NTNU [1]. Oppgaven så på mulighetene for å lage en utviklingsplattform som skal gjøre det enklere å utvikle digitaldesign med FPGAer, som Arduino gjør for mikrokontroller. Hovedmålet for høstprosjektet var å foreta en grundig undersøkelse for å definere kravspesifikasjoner for Dino-plattformen. Oppgaven brukte Arduino som inspirasjon, og så på hvordan de klarte å skape en enkel og brukervennlig utviklingsplattform for mikrokontroller. I oppgaven ble det kartlagt ulike utviklingsverktøy som eksisterer i dag, og prøvd å sette sammen et nytt og enklere verktøy. Dette ble gjort ved hjelp av dybdeintervjuer med fire personer med ulik bakgrunn og kunnskap. Ved hjelp av svarene fra intervjuene, ble det utarbeidet en oversikt over hva som kjennetegner enkle utviklingsverktøy, samt hva som kreves for at et verktøy skal være lærerikt å bruke.

Argumentene som vil bli presentert i dette avsnittet er hentet fra prosjektoppgaven[1]. Intervjuobjektene ble presentert tre ulike metoder for digital design. Første metoden som ble evaluert var grafisk programmering, det vil si å benytte en grafisk representasjon av maskinvarekomponenter og kabler som man kobler sammen til ønsket design. Styrkene til en slik metode er at den er enkel og intuitiv, men på andre siden kan designet fort bli komplisert og uoversiktlig. Den

andre metoden som ble presentert var programmering ved hjelp av maskinvarebeskrivende språk(HDL). Styrkene til denne metoden er at det er denne metoden som benyttes i industrien, og at en slik metode er effektiv og kompakt. Svakhetene vil være at det krever at brukeren kan et maskinvarebeskrivende språk, og at disse maskinvarespråkene ofte er vanskelige å lære seg. Den siste metoden som ble presentert var programmering ved hjelp av vanlige programvarespråk som Python og C++. Fordelene med denne metoden er at programvarespråk ofte er lettere å forstå enn maskinvarebeskrivende språk og at brukeren allerede kanskje kan språket. Ulempen er at det kan være vanskeligere for brukeren å forstå hvordan maskinvare egentlig fungerer siden man ikke benytter dedikert maskinvarebeskrivende språk.

Fra intervjuene viste det seg at grafisk fremstilling av design er viktig for å gi forståelse for hvordan systemet fungerer. Det viste seg også at maskinvareutvikling ved hjelp av koding er den raskeste og mest brukte metoden. Dette resulterte i at Dino er en kombinasjon av disse to løsningene. Dino ønsker å gjøre brukeren i stand til å utvikle designet med koding, og forstå utformingen gjennom en grafisk representasjon. Det ble også påpekt at enkelt og intuitivt brukergrensesnitt er viktig for å god brukeropplevelse.

Kapittel 4

Prototypeutvikling og Programkrav

Denne masteroppgaven er i hovedsak en utviklingsprosess hvor målet er å utvikle en plattform kalt Dino, for å forenkle FPGA-design. Av den grunn er mye av tiden brukt på hvordan man effektivt kan utvikle programvare som klarer å løse problemstillingen man har satt seg. Oppgaven er en forlengelse av tidligere arbeid som ble gjort i en prosjektoppgave fra høsten 2018(kapittel 3.8). Resultatene fra denne oppgaven ligger derfor til grunn for hvordan utviklingen av denne utviklingsplattformen har blitt gjennomført. Først i dette kapittel blir det introdusert hvordan prototyputviklingen har blitt gjennomført, videre blir programkravene presentert og evaluert. Til slutt presenteres intervjuobjektene som har bidratt under evaluering.

4.1 Prototypeutvikling

Å lage programvare fra bunnen av er en omfattende og stor oppgave. Det er en prosess som kan vare i mange år. I denne oppgaven er målet derfor å lage en prototype som inneholder de aller mest nødvendige funksjonene slik man kan se om potensialet eksisterer. Å utvikle en prototype kan også være en omfattende og krevende jobb. Derfor er det en fordel å benytte seg av eksisterende teori om programvareutvikling slik at man forsikrer seg at utviklingen utføres på en effektiv og systematisk måte. Av denne grunn bygger mye av utviklingsprosessen på teorien som er presentert i kapittel 2.3, som handler om effektiv programvareutvikling.

4.2 Programkrav

Det første som gjøres i en utviklingsprosess er å definere programkrav til det endelige produktet. Programkrav inneholder informasjon om funksjonalitet og anvendbarhet til programmet. Hvert krav har også en prioritet som er til stor nytte når man skal begynne utviklingen av en prototype. Programkravene til utviklingsplattformen Dino kommer fra tre ulike faktorer. Dette er prosjektoppgaven(kapittel 3.8), inspirasjon fra Arduino(kapittel 2.5) og til slutt krav som følge av at det er et utviklingsverktøy(kapittel 2.2).

Utviklingen av Dino startet allerede høsten 2018. Da ble det gjennomført flere intervjuer som omhandlet hvordan man enklest og best kan utvikle digitale kretser for FPGA. I denne masteroppgaven brukes derfor disse resultatene som utgangspunkt når programkravene utarbeides. Fra intervjuene som ble resultatet at det var mest hensiktsmessig å benytte maskinvarespråk til å designe digitale kretser. Fra dette betyr det at det trengs en tekstbehandlingsfunksjonalitet, slik at brukeren har mulighet til å skrive kode. Det ble også konkludert med at en grafisk fremvisning av designet er viktig for å øke forståelsen av hvordan digitaldesign fungerer. Til slutt var programmets enkelthet en viktig faktor for god brukeropplevelse.

I tillegg til prosjektoppgaven er denne oppgaven inspirert av Arduino, og det vil derfor være naturlig å se på hva som gjorde at Arduino ble en suksess. Arduino ble nødvendigvis ikke en suksess fordi det er billig eller at de har et kraftig utviklingsverktøy, men heller fordi det er enkelt å bruke. Du kan ha svært lite erfaring med programmering og elektronikk, og fortsatt klare å få en lysdiode til å begynne å blinke. Det eneste brukeren trenger å gjøre er å koble kortet til PC-en, åpne en medfølgende eksempelkode, og deretter trykke på en "kjør" knapp. Kort forklart har Arduino lyktes med å gjøre utviklingen på mikrokontrollere veldig enkelt og intuitivt, og dette er noe som ønskes at kan overføres til Dino. Arduino har også stort fokus på eksempeldesign som gjør det enkelt for uerfarne utviklere å forstå hvordan ulike problemer kan løses.

Kravene som følger av at Dino skal være et utviklingsverktøy, er blant annet muligheten for syntese, plassere og rute, og programmering av brikke. Disse tre funksjonene er avgjørende for at det skal være mulig å kunne overføre et kodet maskinvaredesign til en FPGA. I tillegg til disse, er det flere funksjonaliteter som

kan være en fordel å ha. For å sikre gyldig kode uten syntaks feil eller lignende vil det være en stor fordel med en god feilmeldingsfunksjonalitet. For å øke kvaliteten på designet man lager og for å verifisere at designet gjør det man ønsker at den skal gjøre, vil det også være fordel med mulighet for å simulere designet.

Fra disse tre faktorene sitter vi igjen med følgende krav:

1. tekstbehandling
2. grafisk fremvisning av designet
3. enkelt brukergrensesnitt
4. eksempeldesign
5. syntese
6. plassere og rute
7. programmering av brikke
8. feilmelding
9. simulering

Nå som kravene er satt, trenger vi å tilordne hvert krav en prioritet og beskrive hva salgs type krav det er. Prioriteten vil være til hjelp når vi senere skal lage en plan for prototypen, siden vi da vet hva som regnes som viktigst, og derfor hva prototypen bør inneholde. For å beskrive kravet, skiller man mellom funksjonalitet og anvendbarhet. Funksjonalitet handler om hva slags oppgaver programmet kan utføre, og hva slags egenskaper programmet innehar. Anvendbarhet handler om å øke brukeropplevelsen og brukbarheten.

Vi starter med tekstbehandlingsfunksjonalitet, dette er et funksjonelt krav og kanskje et av de viktigste kravene, og noe som må implementeres for at verktøyet skal kunne fungere. Videre har vi grafisk fremvisning av designet. Dette er også svært viktig, siden Dino har som mål å få brukeren til å forstå digital kretsdesign. Kravet går innunder anvendbarhet siden dette vil hjelpe brukeren med å forstå hva han/hun har designet, og derfor øke brukeropplevelsen. Neste krav er enkelt brukergrensesnitt. Dette går under anvendbarhet og kan være viktig for at brukeren skal få en god opplevelse av verktøyet. Siden dette programmet fokuserer på å være mest mulig anvendbart og intuitivt for brukeren settes denne

til høy prioritet. Videre kommer kravet om eksempeldesign, som også går under kravtypen anvendbarhet. Dette kravet er ikke avgjørende for et godt program, men vil være til stor nytte når det kom til å hjelpe uerfarne utviklere med å starte et nytt prosjekt, allikevel settes dette til middels prioritet. De tre neste kravene; syntese, plassere og rute, og programmering av brikke, er alle funksjonelle krav som er helt nødvendige for at programmet skal bli et utviklingsverktøy. Derfor vil vi kunne gi disse høy prioritet. Feilmelding er neste krav, og dette går under kravtypen anvendbarhet. Kravet har en høy prioritet selv om det ikke er avgjørende for at verktøyet skal fungere. Grunnen til at dette kravet settes til høy er fordi det anses som nødvendig for at brukeren skal forstå hva som er galt hvis en syntetisering ikke klarer å fullføre. Til slutt har vi simulering. Dette er et funksjonelt krav og kan være til god nytte når man skal verifisere at designet fungerer som ønsket. Allikevel settes dette kravet til middels, på grunn av at det kan anses som ikke avgjørende for å produsere et fungerende design. Totalt sett etter denne evalueringen sitter vi igjen med en prioritetsliste som vist i tabell 4.1.

TABELL 4.1: Programkrav

Krav	**Type**	**Prioritet**
Tekstbehandling	Funksjonelt	Høy
Grafisk fremvisning av designet	Anvendbarhet	Høy
Enkelt og intuitivt brukergrensesnitt	Anvendbarhet	Høy
Eksempeldesign	Anvendbarhet	Middels
Syntese	Funksjonelt	Høy
Plassere og rute	Funksjonelt	Høy
Programmering av brikke	Funksjonelt	Høy
Feilmelding	Anvendbarhet	Høy
Simulering	Funksjonelt	Middels

4.3 Intervjuobjektene

Gjennom utviklingen av Dino vil det gjennomføres evalueringer. For å gjennomføre disse, benyttes det eksterne intervjuobjekter. For å få en god kobling mellom prosjektoppgaven og denne masteroppgaven benytter vi oss av samme intervjuobjekter. Dette gjør at valgene som har blitt gjort tidligere i prosjektoppgaven er kjent for intervjuobjektene. De vil være innforstått med oppgavens mål, og de vil få mulighet til å se om sine egne forestillinger om hvordan Dino bør være samsvarer med det endelige resultatet. De som skal evaluere Dino er tre studenter og en professor, hvor alle stiller med ulik kompetanse og erfaring. Den første studenten vil i denne oppgaven bli presentert som S1, student nummer to som S2, student tre som S3 og professor som P1. S1 er en student med ingen erfaring om maskinvaredesign, og har aldri benyttet et maskinvarebeskrivende språk før. S2 er en student innen elektronikk, men har kun hatt enkelte fag som har gått i overflaten på temaet. Studenten anser sin kunnskap som liten, men forstår hovedideene. Den siste studenten S3, studerer elektronikk og har mye erfaring på feltet. Studenten har erfaring med både Verilog, VHDL og SystemVerilog. Til slutt har vi professoren P1 som underviser på feltet, og har mye erfaring med maskinvareprogrammering, samt mye kunnskap om hvordan å lære bort maskinvareprogrammering.

Kapittel 5

Første prototype: Modell

Etter at programkravene er bestemt kan man starte å realisere disse gjennom prototyper. Når man starter utviklingen av en prototype vet man ennå ikke hvordan denne prototypen vil være, eller hva den vil inneholde. For å få svar på dette gjennomføres en fire fases prosess. Dette kapittelet tar for seg hver av disse fasene for første prototype. Valgene som blir gjort underveis blir argumentert for og beskrevet her.

5.1 Plan

Når programkravene er definert kan det utarbeides en plan for prototypen. Som nevnt i kapittel 2.3.1 handler dette om å se på programkravene og velge ut noen av kravene som man ønsker å verifisere gjennom en prototype. I tillegg til dette lages det en programflyt eller det isenesettes et scenario som beskriver problemet prototypen skal løse. Til slutt spesifiseres innholdet i prototypen og hvor likt dette innholdet er i forhold til det endelige produktet. Grunnen til at man lager en slik plan er for å forsikre seg om at prototypen man lager faktisk vil sjekke de funksjonene og egenskapene man ønsker.

Første steg er som nevnt å velge ut ønskede krav. Siden dette blir den første prototypen, er det naturlig å velge ut de viktigste kravene, samt de kravene som vi er mest usikre på. Fokuset for denne prototypen er mer på anvendbarheten enn på funksjonaliteten. Vi velger de kravene fra programkravene(tabell 4.1) som virker å være viktigst for å demonstrere ideen bak Dino. I tillegg er det en fordel å ha et oversiktlig brukergrensesnitt, så dette er også naturlig å legge inn i prototypekravene. Da ender vi opp med en liste av prototypekravene som presentert i tabell 5.1.

TABELL 5.1: Protoypekrav

Krav	**Type**	**Prioritet**
Tekstbehandling	Funksjonelt	Høy
Grafisk fremvisning av designet	Anvendbarhet	Høy
Enkelt og intuitivt bruker-grensesnitt	Anvendbarhet	Middels
Feilmelding	Anvendbarhet	Høy

Videre skal vi lage en programflyt eller et scenario. I dette tilfelle velger vi å lage et scenario. Ved å lage et scenario blir det lettere å beskrive målet med denne prototypen. Å lage et scenario betyr å lage en oppgave eller aktivitet som dekker alle de viktigste funksjonalitetene som bør være inkludert i modellen. Scenarioet vi kan se for oss er en bruker som har liten erfaring med digital kretsdesign og som ønsker å programmere sin første FPGA. Brukeren har et ønske om å programmere ved hjelp av et maskinvarebeskrivende språk, men trenger hjelp til å komme i gang. Personen som skal bruke verktøyet er ikke helt sikker på hvordan digital design egentlig fungerer.

Etter at prototypekrav og et scenario er bestemt, må vi spesifisere innholdet i prototypen og tilordne trofasthet til hver av dem. For å kunne jobbe videre etter denne prototypen vil det være en fordel å være sikker på at layout på brukergrensesnittet er som ønsket, det vil derfor være smart å tilordne dette innholdet høy trofasthet. Et annet element som også må være en del av et utviklingsverktøy er filsystemet. Brukeren må ha mulighet til å kunne lage nye filer, eller åpne eksisterende. Ser vi på hva som er målet med Dino(kapittel 1.2), så er dette å lage et verktøy som er lett å bruke for personer med liten erfaring innen digital design, og at verktøyet skal hjelpe brukeren med å forstå sitt eget design. Derfor bør prototypen også fokusere på den grafiske fremstillingen av designet, samt å gi god tilbakemelding på feil i kode, disse egenskapene vil ha en trofasthet på middels. Grunnen til det er at det endelige produktet kan avvike fra hva som blir presentert i denne prototypen. Fokuset her er heller at elementene er tilstede og det som presenteres gir en ide om mulighetene. Listen over innholdet og trofasthetene er presentert i tabell 5.2.

TABELL 5.2: Innhold og trofasthet

Innhold	Trofasthet
Tekstbehandling	Middels
Grafisk fremvisning av designet	Middels
Enkelt og intuitivt bruker-grensesnitt(layout)	Høy
Eksempeldesign	Middels
Filsystem	Lav
Feilmelding	Middels

5.2 Spesifikasjoner

I forrige fase ble det bestemt hva prototypen skal innholdet. I denne fasen bestemmes hvordan man skal lage protypen. Det første som gjøres i denne fasen er å bestemme seg for egenskapene for prototypen. Som beskrevet i kapittel 2.3.2 bestemmes følgende punkter; målgruppe, stadium, hastighet, levetid, uttrykk, stil, medium og trofastheten for prototypen. Når alle disse punktene er evaluert velges en passende prototype metode. Man har gjennom alle de foregående fasene opparbeidet seg en grundig oversikt over hva som kreves av prototypen, valget som derfor utføres er godt gjort rede for.

Det første steget i denne fasen er å spesifisere egenskapene ved prototypen. Vi organiserer egenskapene i en liste på lik måte som i tabell 2.1 fra kapittel 2.3.2. Som beskrevet i det kapittelet har hver egenskap mellom to og tre valg. For denne prototypen velger vi oss egenskapene som beskrevet i tabell 5.3. For målgruppe blir det nødvendig å velge eksterne personer, siden denne oppgaven utføres av kun en person. Stadiet i utviklingen av Dino kan anses å være i en midtfase siden mye arbeid allerede har blitt utført gjennom prosjektoppgaven(kapittel 3.8). Hastigheten på utviklingen av denne prototypen ønsker vi at skal være rask. Levetiden setter vi til middels lang, og uttrykket velger vi til å være konseptuell. Stilen på denne prototypen ønsker vi skal være interaktiv, slik at vi også får sjekket de funksjonelle kravene. Vi benytter fysisk medium, og trofastheten til hele prototypen settes til middels nivå.

TABELL 5.3: Egenskaper for prototype

Egenskaper		**Valg**
Mål-gruppe	Intern	
	Ekstern	x
Stadie	Tidlig	
	Midt	x
	Sen	
Hastighet	Rask	x
	Flittig	
Levetid	Kort	
	Middels	x
	Lang	
Uttrykk	Konsept	x
	Eksper	
Stil	Fortelling	
	Interaktiv	x
Medium	Fysisk	x
	Digital	
Trofasthet	Lav	
	Middels	x
	Høy	

Etter at prototypeegenskapene er bestemt, kan vi begynne å se på hvilken prototypemetode som passer. For å gjøre dette valget kan vi benytte svarene vi fikk i tabell 5.3 og sammenligne disse med tabell 2.1(kapittel 2.3.2) som viser korrelasjon mellom prototypeegenskaper og prototypemetode. Ser vi på tabell 2.1 er det tre ulike metoder som ser ut til å kunne passe; papir, digital, video og Wizard of Oz. For denne prototypen benytter vi oss av papir metoden. Grunnen for det er at dette er en enkel metode, og som er raskere å gjennomføre enn de andre metodene. Denne prototypen kommer ikke til å inneholde fysiske endringer i modellen ved brukerinteraksjon, av den grunn faller digital, video og Wizard of Oz bort, siden alle disse er ment å vise disse endringene.

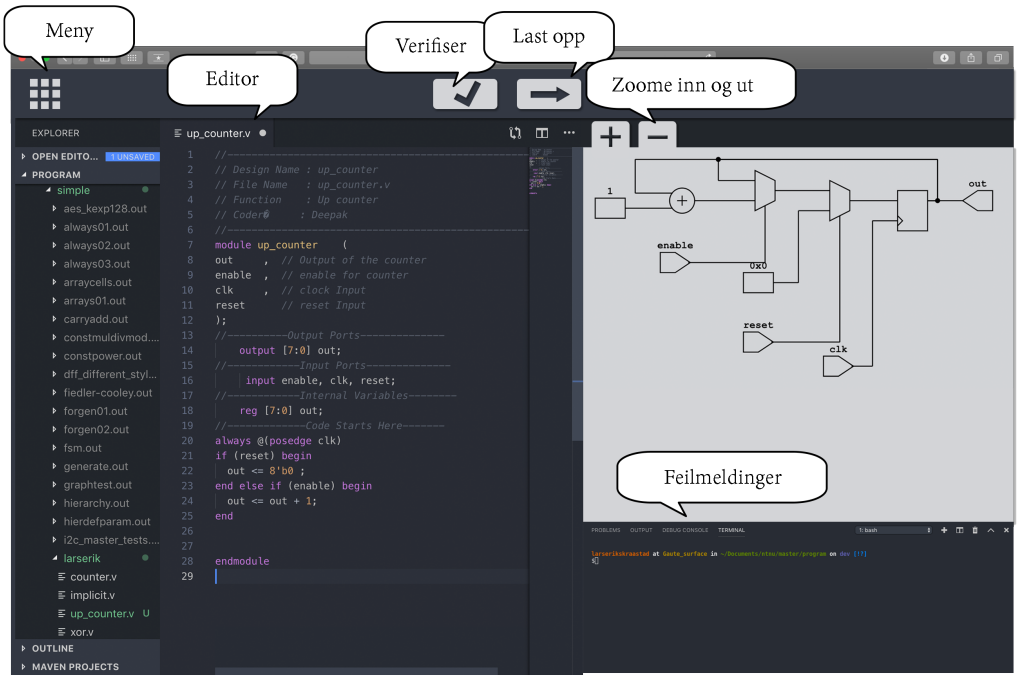
Nå som vi har valgt prototypemetode, kan vi bestemme oss for hva slags verktøy som bør benyttes for å realisere prototypen. Her velger vi mellom 8 ulike verktøy. Verktøyet som velges er bestemt av hvilken metode som ble valgt i avsnittet over. Tabell 2.2 i kapittel 2.3.2 gir en oversikt over hvilket verktøy som passer til hvilken metode. Siden vi for denne prototypen endte opp med en papirmodell vil eksempler på verktøy være Word, Papir, Excel og Photoshop. Alle disse verktøyene einer seg til å lage en papirmodell, så hvilken man velger kommer an på sine egne ferdigheter og preferanser. For denne prototypen benytter vi oss av Photoshop, på grunn av verktøyets funksjonaliteter.

5.3 Designe

I denne fasen lages prototypen. Det som må gjøres før man starter på utviklingen er å få en oversikt over retningslinjer for godt design. Ved å gjøre dette forsikrer man seg at brukergrensesnitt og visuelt design følger gode standarder. En liste med gode retningslinjer er presentert i tabell 2.3 i kapittel 2.3.3. Når disse retningslinjene er evaluert kan designet realiseres.

Vi starter derfor med se på retningslinjene i tabell 2.3, og bestemmer oss for hva som er viktigst for denne papirmodellen. Starter vi med det visuelle, så vil informasjonsflyt være en viktig faktor. Altså hvor vil øynene til brukeren bevege seg. Av prototypekravene som vi bestemte oss for tidligere i dette kapitelet, vil de visuelle elementene være tekstbehandlingsfeltet, den grafiske fremvisningen og feltet for feilmeldinger. Programflyten vil være; brukerinput i tekstbehandlingsfeltet, koden vil så kompileres og vi får enten en feilmelding eller en grafisk fremvisning. Dette betyr at øynene vil naturlig bevege seg fra tekstbehandlingsfeltet til enten feilmelding eller en grafisk fremvisning, som betyr at disse to elementene bør ligge i nærheten av hverandre. Med samme argument vil retningslinjen om logisk gruppering også være oppfylt. Feilmelding og grafisk fremvisning bør være gruppert. Neste steg vil være å se på kategorien brukergrensesnitt. Her vil retningslinjen om lærbarhet og snakke publikums språk være de to viktigste retningslinjene. Brukeren har liten erfaring med digital design og utviklingsverktøy, det vil derfor være smart å unngå avanserte uttrykk og funksjoner, og heller fokusere på enkelhet. I tillegg til disse retningslinjene fokuserer vi også på å minimere kognitiv belastning ved for eksempel å benytte ikoner istedenfor tekst. Dette vil

også føre til økt universell tilgjengelighet, ved at språkbarriere og lignende vil reduseres.



FIGUR 5.1: Papirmodell

Etter at vi nå har tatt for oss retningslinjer for designet, kan vi starte å realisere modellen. Designet for denne prototypen er presentert i figur 5.1. Vi starter med det viktigste elementet, som vil være tekstbehandlingsfeltet. Dette plasserer vi mest mulig i senter, da dette er feltet som brukeren bruker mest. Videre grupperer vi feltene for grafisk fremvisning og feilmelding. For å gjøre hele brukeropplevelsen mest mulig kjent, benytter vi oss av et mye brukt redigeringsprogram. Vi velger derfor å bruke Visual Studio Code[58] for dette.

5.4 Evaluere

Prototypen er nå laget og vi må evaluere resultatet. For at evalueringen skal bli best mulig er det viktig å lage en god plan slik at man er godt forberedt. Det er

også viktig å informere de som skal evaluere modellen godt, slik at de har en mest mulig korrekt forventing om hva som møter dem.

Vi starter derfor med å lage en plan for hvordan evalueringen skal gjennomføres. Først setter vi forventningene. For denne modellen vil det være at vi befinner oss i midten av utviklingen av Dino, vi har en ide om hvordan designet vil være, men har ennå ikke laget en interaktiv modell. Målet med det endelige programmet vil være å ha et verktøy som gjør design av digitale kretser på FPGA enkelt, også for personer med liten erfaring.

Med en slik introduksjon vet personen som skal evaluere modellen hva målet er, og hvor i prosessen vi befinner oss. Videre må vi bestemme oss for hva vi ønsker å få svar på. For denne modellen ønsker vi å få svar på følgende spørsmål.

- Hva liker du best/minst ved designet?
- Hvor enkelt virker verktøyet?
- Hvilken funksjonalitet virer mest lovende?
- Hva føler du mangler?

Ved å stille disse spørsmålene skal vi være godt rustet til å fortsette utviklingen av Dino. Med spørsmålene som er beskrevet over vil det mest egnede verktøyet for denne evalueringen være en brukertest, i form av en presentasjon av modell og ønsket funksjonalitet.

5.5 Resultat

Fra brukertesten fikk vi en rekke svar. Det var en felles enighet om at den grafiske fremvisningen var god, og at programmet som en helhet virket veldig enkelt. At brukeren ved kun ett trykk vil få presentert kretsen grafisk og med et trykk til så vil koden bli lastet opp til kortet, var positivt. Det som virket mest lovende var i følge intervjuobjektene den visuelle fremvisningen, og hvordan brukeren enkelt kunne se sammenheng mellom kode og fysisk design. S1 likte veldig godt knapper istedenfor tekst og mange menyer. S2 trakk frem layout på programmet som den største styrken. Professor P1 syntes designet var smakfult og logisk organisert. Når det kom til forbedringspotensialet, var det ønske fra S2 og S3 om å

variere graden av tilgjengeligheten av avanserte funksjoner. Altså, hvis brukeren ønsker å utføre avanserte endringer på designet, så burde disse funksjonalitetene være synlige hvis man ønsker. S3 hadde også et ønske om å ha mulighet for simulering av design, og ved hjelp av å highlighte kode og de grafiske komponentene, vises flyten til designet. I tillegg til dette var det viktig for intervjuobjektene at tilbakemeldingene man får bør være gode, slik at det er lett å forstå hva som er galt med designet. Til slutt var det et ønske om at effektiviteten til koden kunne bli vist.

Kapittel 6

Andre prototype: Kodet

Etter å ha designet første prototype vil neste steg være å realisere modellen som er vist i Figur 5.1. Vi er nå sikre på hvordan designet bør være, men vi trenger å implementere funksjonaliteten. For å gjøre dette følger vi samme fremgangsmåte som for første prototype. Vi starter med en plan, deretter spesifiserer vi innholdet, så designes prototypen før den til slutt evalueres. Denne prototypen vil være den siste som utvikles i denne masteroppgaven, og mye av tiden har gått med på implementering og utvikling av denne prototypen. Av den grunn vil dette kapittelet ha et større fokus på designfasen i motsetning til forrige prototype.

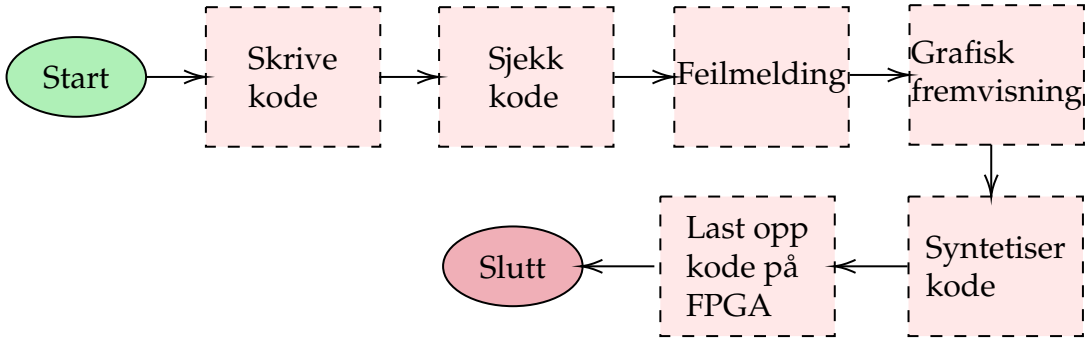
6.1 Plan

Vi starter som nevnt først med å lage en plan. Når vi lager en plan, begynner vi med å bestemme kravene til prototypen. Denne gangen ønsker vi å implementere funksjonalitet, og kravene vil av den grunn være mer rettet mot dette enn anvendbarheten. Vi velger de kravene fra programkravene (tabell 4.1) som fokuserer på funksjonalitet, og som kreves for å få demonstrere potensialet til Dino. Vi ender da opp med prototypekravene som er vist i tabell 6.1

Videre skal vi lage et scenario. Siden vi i denne prototypen ønsker å se på funksjonalitet og hvordan hele programmet virker i sin helhet, vil det være naturlig å lage et mer omfattende scenario. Vi tar utgangspunkt i scenarioet fra forrige prototype. Den generelle programflyten for denne prototypen er vist i figur 6.1. Brukeren setter seg ned foran PC-en og skal lage sitt første FPGA design. Personene har liten erfaring med digital design, så når han/hun åpner Dino i nettleseren får han/hun presentert en mal for hvordan man skal starte å skrive maskinvare. Personen ønsker å skrive minst mulig kode, og velger derfor å åpne

TABELL 6.1: Protoypekrav

Krav	**Type**	**Prioritet**
Tekstbehandling	Funksjonelt	Høy
Grafisk fremvisning av designet	Anvendbarhet	Høy
Enkelt og intuitivt brukegrensesnitt	Anvendbarhet	Middels
Eksempeldesign	Anvendbarhet	Middels
Syntese	Funksjonelt	Høy
Simulering	Funksjonelt	Lav
Plassere og rute	Funksjonelt	Høy
Programmering av brikke	Funksjonelt	Høy
Feilmelding	Anvendbarhet	Høy



FIGUR 6.1: Programflyt

et eksempeldesign. Etter at dette eksempeldesignet er lastet opp trykker personen på verifiseringsknappen. Designet blir syntetisert og behandlet i bakgrunnen og brukeren får fremvist en grafisk fremstilling av designet, eller en feilmelding hvis designet inneholder feil. Personen kan nå se på det visuelle designet og verifisere for seg selv at dette er slik han/hun ønsker at designet skal være. Er personen fornøyd, kobler han/hun FPGAen inn i PC-en. Deretter trykker han/hun på last opp knappen. Den syntetiserte koden lastes opp og kjører nå på den tilkoblede FPGAen.

Nå som vi har oversikt over målet og kravene til denne prototypen, vil det være nødvendig å spesifisere innhold og trofasthet. Dette er vist i tabell 6.2. Det

TABELL 6.2: Innhold og trofasthet

Innhold	**Trofasthet**
Tekstbehandling	Middels
Grafisk fremvisning av designet	Middels
Eksempeldesign	Lav
Syntese	Middels
Plassere og rute	Middels
Enkelt og intuitivt brukergrensesnitt(layout)	Høy
Filsystem	Lav
Programmering av brikke	Middels
Feilmelding	Middels

visuelle ble demonstrert i den første prototypen vi lagde, men vi velger allikevel å beholde disse i denne prototypen også. Grunnen til det er at vi ønsker å vise hvordan totalopplevelsen med programmet vil være. Trofastheten til de ulike visuelle elementene vil settes til middels, mens layout velger vi til å være høy. Vi trenger også å implementere et filsystem, denne er satt til lav da implementeringen av dette mest sannsynlig kommer til å endres i et ferdig produkt. Funksjonalitet relatert til behandling av kode, setter vi til middels. Da disse verktøyene enkelt kan bli byttet ut med andre eksisterende løsninger.

6.2 Spesifikasjoner

Etter at vi har definert krav, scenario og trofasthet til innholdet kan vi starte å spesifisere egenskapene. Etter at egenskapene er bestemt, velger vi en prototypemetode som passer. Til slutt ser vi på hvilket verktøy som egner seg for denne type prototypemetode.

På samme måte som for første prototype starter vi med å evaluere følgende punkter; målgruppe, stadium, hastighet, levetid, uttrykk, stil, medium og trofastheten for prototypen. Som for papirmodellen vil målgruppen for denne prototypen være eksterne personer, siden det ikke eksisterer noen interne. Vi har siden

TABELL 6.3: Egenskaper for prototype

Egenskaper		**Valg**
Mål-gruppe	Intern	
	Ekstern	x
Stadie	Tidlig	
	Midt	
	Sen	x
Hastighet	Rask	
	Flittig	x
Levetid	Kort	
	Middels	
	Lang	x
Uttrykk	Konsept	
	Eksper	x
Stil	Fortelling	
	Interaktiv	x
Medium	Fysisk	
	Digital	x
Trofasthet	Lav	
	Middels	
	Høy	x

forrige gang forflyttet oss fra en midt-fase til sen-fase. Siden vi ønsker å implementere mange funksjonelle krav, og at det meste av innholdet har en trofasthet på middels til høy, vil det være naturlig å velge en flittig utviklingshastighet. Prototypen ønsker vi at skal ha lang levetid, og uttrykket velger vi at skal være eksperimentell. Siden vi skal kombinere et godt brukergrensesnitt med en rekke funksjonaliteter ønsker vi at prototypen skal ha en interaktiv stil. Videre ønsker vi en digital prototype med en samlet trofasthet på høy. Valgene er presentert i tabell 6.3.

Vi trenger nå å bestemme hvilken prototypemetode som egner seg for disse egenskapene. Sammenligner vi resultatene som vi fikk (tabell 6.3) med tabell 2.1 er det to metoder som passer; digital eller kodet prototype. En digital prototype

er ganske lik en papirmodell, bortsett fra at brukerinteraksjoner gir en fysisk endring i modellen(kapittel 2.3.2). Siden vi ønsker en prototype som er enda nærmere det endelige produktet velger vi derfor kodemetoden.

Videre må vi velge et verktøy for å realisere prototypen. Siden valget endte på en kodemetode vil det være naturlig å velge et programvareverktøy. Når man velger verktøy vil det være en fordel å velge et verktøy man er komfortabel med, av den grunn velger vi Visual Studio[58].

6.3 Design

Nå som planen er laget, og spesifikasjonene for prototypen er satt, kan vi begynne utviklingen. Før vi setter i gang med dette må vi få en oversikt over retningslinjene for godt design. Ved å gjøre dette forsikrer man seg at brukergrensesnitt og visuelt design følger gode standarder. En liste med gode retningslinjer er presentert i tabell 2.3 i kapittel 2.3.3.

Starter vi å se på retningslinjene for godt design, ser vi at mange av disse allerede ble benyttet under utviklingen av første prototype. Det betyr at, så lenge vi benytter den modellen som utgangspunkt så er mange av retningslinjene allerede oppfylt.

For å realisere alle kravene som vi har satt for denne prototypen vil det være hensiktsmessig å ikke utvikle alle kravene på en gang. Av den grunn deler vi opp utviklingen i flere faser. Først lager vi det generelle brukergrensesnittet og strukturen til programmet, før vi senere legger til flere funksjonaliteter.

Når vi skal lage et nettbasert program er det vanlig å dele programmet opp i frontend[1] og backend[2]. Frontend er en viktig del av Dino og kan være avgjørende for om verktøyet lykkes i å nå sitt mål om å være intuitivt, samt at det oppfordrer til læring. I kapittel 5 utarbeidet vi en papirmodell(figur 6.2a), som videre ble evaluert av flere eksterne intervjuobjekter. Intervjuobjektene var som nevnt positive til brukergrensesnittet. Av den grunn blir første del i dette steget å programmere

[1]Frontend er den visuelle delen av et program[59]

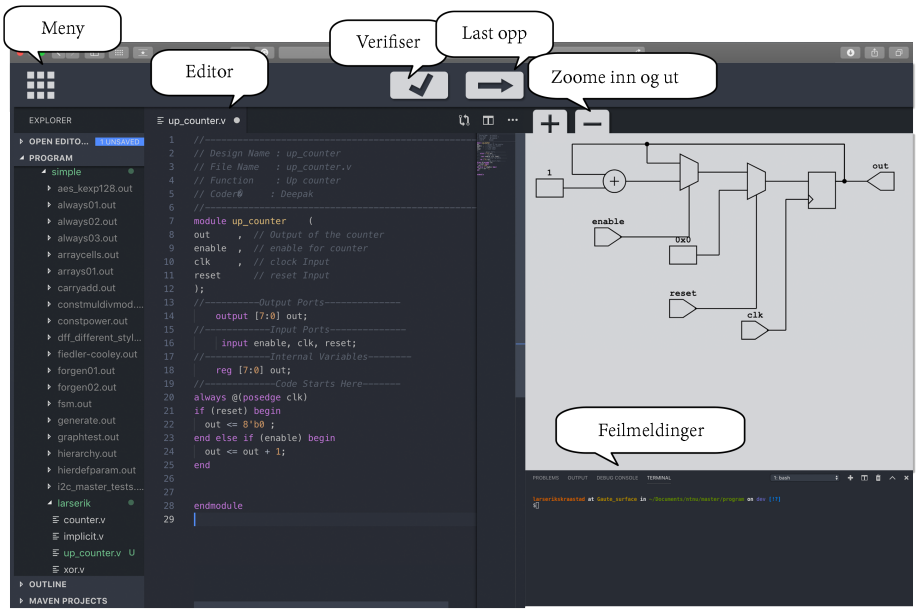
[2]Backend er den delen av et program som tar for seg de tunge kalkuleringene, og er den delen som brukeren ikke ser[60]

brukergrensesnittet til denne modellen(Figur 6.2a).

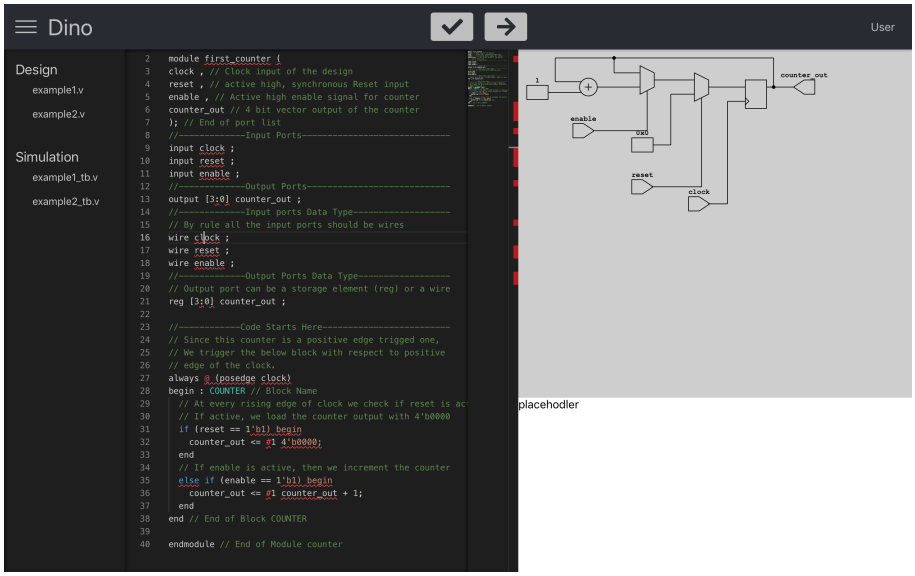
6.3.1 Første utvikling: Brukergrensesnitt

Når man lager en nettside skriver man i HTML, CSS og JavaScript(kapittel 2.4). Disse tre språkene brukes for å beskrive innholdet, utseende og oppførselen til nettsiden. For å utvikle frontend til Dino, ønsker vi å benytte et rammeverk som vil hjelpe oss i organiseringen og effektiviteten i utviklingen. Rammeverket organiserer HTML, CSS og JavaScript på ulike måter, for å forbedre utviklingsprosessen. For Dino sitt vedkommende benytter vi React[61]. React er et bibliotek som har fokus på en-sides nettsider som er bra for datahåndtering og raske dataendringer. Dette er noe som vil passe Dino godt, da mye data vil sendes mellom frontend og backend, og en rask respons er en fordel for å øke brukeropplevelsen.

I denne første programmerte utviklingen av Dino skal vi kun fokusere på å få designet riktig. Vi bruker figur 5.1 som referanse. Det første vi gjør er å organisere nettsiden med tomme bokser. Ved å gjøre det slik får vi laget layoutet, og det vil da bare være å fylle inn de elementene vi måtte ønske. Etter at layoutet er laget, starter vi med å lage navigeringsfeltet helt øverst. Vi lager menyen helt til venstre, plasserer to knapper på midten og lager et brukerfelt helt til høyre. Videre tar vi for oss tekstbehandlingsfeltet(editor). Det finnes mange tekstbehandlingsverktøy fra før, derfor vil det være en god løsning å benytte noe som allerede eksisterer. Det er i hovedsak to ulike tekstbehandlingsverktøy som vurderes. Enten Monaco[62], som er tekstbehandlingsverktøyet som Visuel Studio Code[58] benytter, eller så kan man velge Theia[63]. Theia er et mer omfattende verktøy som har en rekke avanserte innebygde funksjonaliteter. Theia er organisert slik at man enkelt kan legge til ønskede funksjonaliteter. Hvis man velger denne løsningen så vil vi ende opp med å måtte benytte Theia som hovedplattform, og at vi videre kunne lagt til de funksjonaliteten vi måtte trenge. Dette er en løsning som vi ikke ønsker siden vi ønsker mer frihet når det kommer til brukergrensesnitt, og vi velger derfor å bruke Monaco.



(A) Papirmodell



(B) Første kodet prototype

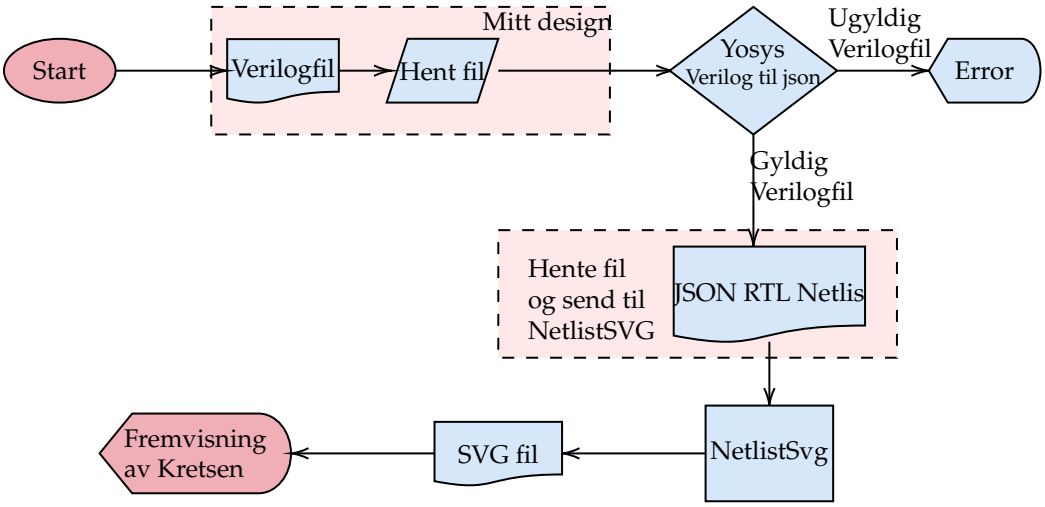
FIGUR 6.2: Sammenligning av papirmodell og første kodet prototype

Først implementerer vi Monaco tekstbehandlingsverktøy, og et navigeringsfeltet øverst på siden. Videre lager vi et filsystem som plasseres til venstre på siden. Filsystemet er bare en liste med tilgjengelige filer. Neste implementering er feltet for grafisk fremvisning. Den grafiske fremvisningen er et SVG bilde. Så vi lager et grått felt som kan lese SVG filer. Til slutt legger vi inn et lite felt hvor feilmeldinger skal bli presentert. Figur 6.2 viser både første prototype og nåværende design, for sammenligning.

Filstrukturen til det endelige frontendet er presentert i appendiks A, og koden som er utviklet ligger i vedlagt zip-fil.

6.3.2 Andre utvikling: Behandle Verilog-kode og generere SVG-fil

Nå som vi har laget basen for frontend, trenger vi å se på det som skal foregå i backend. Vi skal i denne utviklingen lage den delen av programmet som skal behandle Verilog-koden, og generere en SVG fil av kretsen. For å gjøre dette benytter vi to ulike rammeverk; Yosys(kapittel 3.3.1) og Netlistsvg(kapittel 3.4). Først har vi Yosys som er et synteseverktøy som konverterer oppførsel basert Verilog kode til RTL kode(kapittel 3.3.1). Denne RTL koden vil Yosys deretter eksportere til et JSON objekt. For å generere en SVG fil av kretsen benytter vi oss av Netlistsvg. Netlistsvg tar inn JSON objektet og tegener en skjematisk fremstilling av kretsen og lagrer filen i SVG format



FIGUR 6.3: Yosys og NetlistSvg programflyt

KODE 6.1: Verilog til JSON med Yosys[49]

```
yosys -p "prep; write_json output.json" input.v
```

KODE 6.2: Json til SVG med Netlistsvg[49]

```
netlistsvg input_json_file -o output_svg_file
```

Flyten til en slik løsning er illustrert i Figur 6.3. For å realisere denne løsningen er det først behov for et skript som kan hente en Verilog-fil og videresende denne filen til Yosys. For å gjøre dette benytter vi oss av kommandoen i kode 6.1. Er designet i Verilog-filen ugyldig vil Yosys returnere error, hvis ikke så returneres en RTL netlist i form a et JSON objekt. Videre trenger vi et skript som henter denne filen og kaller på Netlistsvg, som så returnerer en SVG fil. Dette gjøres med kommandoen som vist i kode 6.2. Til slutt trenger vi å vise denne SVG-filen i nettleseren.

En viktig bemerkning fra denne implementeringen er hvordan Yosys håndterer ugyldig kode. Hvis man sender en ufullstendig eller ugyldig kode til Yosys, vil verktøyet avslutte og returnerer feil. Problemet er at verktøyet ikke forklarer hva som var feil, og hva som må endres i Verilog-koden for å få et gyldig design. Dette er en stor svakhet og noe som må forbedres senere.

6.3.3 Tredje utvikling: Generell backend

Siden Dino er en nettbasert utviklingsplattform trenger vi at frontend (brukergrensesnittet) kommuniserer med backend. Vi lager derfor i den tredje delen av utviklingen, et generelt backend som kommuniserer med brukergrensesnittet vi utviklet i første del av utviklingen. Når vi skal lage backend, må vi først bestemme oss for hvilke programmeringsspråk vi skal benytte. Hvis Dino blir et fungerende program, og man ser potensialet i det, er det tenkt at dette verktøyet kan bli en del av IDE8(kapittel 3.7). IDE8 er utviklet i programmeringsspråket Go[3], og hvis Dino en gang i fremtiden også skal integreres er det en fordel at dette verktøyet også er utviklet i samme språk. Derfor vil backend-delen av Dino bli utviklet i Go.

Når frontend og backend kommuniserer gjøres dette via HTTP(kapittel 2.4.3). Siden Dino er et utviklingsverktøy, vil det være mye data som skal sendes mellom

[3]Et programmeringsspråk utviklet av Google[64]

frontend og backend. Derfor benytter vi oss av protokollen websocket(kapittel 2.4.4) som muliggjør toveiskommunikasjon, og gjør dataoverføring mellom frontend og backend enklere. I tillegg til at vi lager en backend med websocket protokoll, lager vi også en distributør som håndterer alle tilkoblingene til denne nettapplikasjonen. Denne distributøren lager vi slik at koden som en bruker skriver i teksteditoren til Dino, blir koblet til denne bruker i sanntid. Dette gjør at man kan ha flere vinduer oppe til samme tid og dataen vil synkroniseres mellom dem.

TABELL 6.4: Meldinger fra frontend

Type	**Innhold**
Oppstart	Brukerinformasjon
Lagre	Kode som skal lagres
Verifisere	Kode som skal sjekkes for feil og syntetiseres
Prog	Kode som skal overføres til FPGA
Få innhold	Innholdet til filen
Ny fil	Koden til den nye filen

TABELL 6.5: Meldinger fra backend

Type	**Innhold**
Oppstart	Alle tilgjenglige filer
SVG	Bildet av krets
Errormsg	Feilmeldinger
Kode	Koden som er skrevet
Lagre	Verifiserer at kode er lagret i backend
Ny fil	Koden til den nye filen

Det neste vi gjør er å lage funksjonalitet til å håndtere meldinger som kommer fra frontend(nettsiden). Dette kan være meldinger om syntetisering, programmering av brikke, eller lagring av kode. Alle de ulike meldingene som sendes mellom frontend og backend er beskrevet i tabell 6.4 og 6.5. Den første meldingstypene vi lager støtte for er "ny fil". Denne funksjonen tar inn filnavn og innholdet til den nye filen, og lagrer den i en mappe. Videre lager vi støtte for "lagre", som også tar inn filnavn og innholdet, og skriver innholdet til den bestemte filen. Neste funksjon er "få innhold", som tar inn et filnavn og returnerer innholdet til filen. Vi lager også støtte for "oppstart" som går gjennom mappen med lagrede filer og returnerer alle filnavnene, slik at disse kan sendes til frontend og brukeren ser sine tilgjengelige filer.

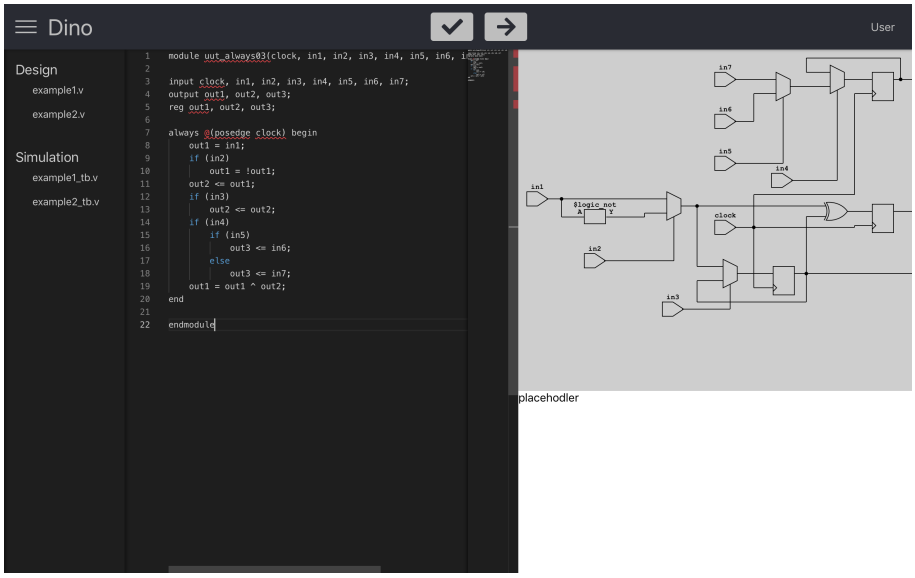
6.3.4 Fjerde utvikling: Ferdigstille og forbedre

I forrige utvikling lagde vi et generelt backend, som kommuniserte med frontend via websocket. Vi lagde støtte for meldingstypene som var relatert til filhåndtering. I fjerde utvikling ferdigstiller vi prototypen og gjør de nødvendige forbedringene som må gjøres for å øke brukeropplevelsen.

For å ferdigstille denne prototypen er det noen funksjonaliteter vi trenger å legge til. Vi må integrere andre utvikling("Behandle Verilogkode", kapittel 6.3.2) med tredje utvikling("Generell backend", kapittel 6.3.3). Siden behandlingen av Verilog-kode kun genererer et SVG bilde av kretsen, trenger vi også å implementere en fullverdig syntetisering, plassering og ruting og programmering til en spesifikk FPGA. Det bør også implementeres et eksempeldesign og kode-mal som brukerne kan bruke som utgangspunkt.

Når det kommer til hva som bør forbedres, er det deler ved brukeropplevelsen som må endres. Det er i hovedsak to viktige funksjoner som mangler for å øke brukeropplevelsen. Den første funksjonen som vi trenger å legge til er muligheten til å navigere seg rundt i bildet av kretsen (zoome ut/inn og flytte rundt). I tilfeller hvor man jobber med mer omfattende design, vil man kunne oppleve at bildet av kretsen blir for stort og hele kretsen vil ikke synes. Dette er illustrert i Figur 6.4.

Den andre viktige funksjonen som mangler, er muligheten for å få feilmelding ved feil i kode. Nåværende prototype generere ingen feilmelding når man prøver å behandle ugyldig kode, som beskrevet i slutten av kapittel 6.3.2. Dette er en stor svakhet, og noe som må implementeres. Verktøyet skal være et hjelpemiddel for



FIGUR 6.4: Et eksempel på en for stor krets

personer med liten eller ingen erfaring, og da er det helt avgjørende at brukeren får den hjelpen han/hun måtte trenge.

Vi starter med å ferdigstille backend. Her trenger vi en fullstendig Verilog til FPGA støtte. Vi begynner med å integrere løsningen fra andre utvikling. Dette gir oss Verilog til bilde av krets. For å kunne lage en fullstendig Verilog til FPGA støtte må vi først bestemme oss for hvilken FPGA vi skal benytte. For denne prototypen velger vi oss et lite og enkelt utviklingskort kalt iCEstick Evaluation Kit med en iCE40 FPGA[65]. Når vi nå skal programmere denne trenger vi å utføre en syntetisering som er rettet mot denne FPGAen. Med Yosys benytter vi kommandoen i kode 6.3.

KODE 6.3: Yosys syntese for iCE40 FPGA

```
yosys -p synth_ice40
      -blif blif/filename.blif
       fileFolder/fileName.v
```

Dette gir oss en blif fil som vi kan benytte når vi skal plassere og rute. Denne prosessen utføres av verktøyet arachne-pnr(kapittel 3.3.2). Arachne-pnr tar inn en pcf fil som forteller hvordan FPGAen er koblet opp på utviklingskortet, og en blif fil som er syntetisert kode fra Yosys, og det returneres en asc fil. Kommandoen for dette er vist i kode 6.4

KODE 6.4: Arachne-pnr plasser og rute kommando

```
arachne-pnr     -d 1k -P tq144
                -o asc/fileName.asc
                -p pcf/icestick.pcf
                blif/fileName.blif
```

Til slutt skal vi programmere FPGAen. For å programmere en FPGA trenger man en binær fil(bin), og siden vi nå sitter med asc fil fra arachne-pnr trenger vi å konvertere denne til bin. Dette gjør vi ved hjelp av Icepack som er en del av IceStorm prosjektet(kapittel 3.3.3). Kommandoen som vi benytter er vist i kode 6.5.

KODE 6.5: Icepack konverterer asc fil til bin fil

```
icepack     asc/fileName.asc
            bin/fileName.bin
```

Når vi nå har en bin fil lastes denne opp til FPGAen ved hjelp av Iceprog som også er et verktøy fra IceStorm prosjektet(kapittel 3.3.3). Kommandoen for dette er vist i kode 6.6.

KODE 6.6: Programmering av FPGA med Iceprog

```
iceprog  bin/fileName.bin
```

Nå som vi har en fullstendig Verilog til FPGA støtte, er neste steg å forbedre brukeropplevelsen ved å implementere en mer omfattende SVG fremviser. Som beskrevet tidligere ønsker vi å ha muligheten til å navigere oss rundt i bildet. Dette går ikke med nåværende løsning da dette kun er en statisk fremvisning av SVG filer. For å løse problemet integrere vi en allerede eksisterende SVG fremvisnings komponent som heter "react-svg-pan-zoom"[66]. Denne komponenten har støtte for alle de funksjonene som vi manglet i den første løsningen. Den siste forbedringen vi trenger å gjøre er fremvisningen av feilmeldinger. For å løse dette problemet, må vi starte i backend. Tidligere i "Tredje utvikling: Generell backend" så vi svakheten til Yosys når det kom til tilbakemelding på kodefeil. Vi må derfor integrere et nytt verktøy for å utføre denne jobben. Et verktøy som har god kode sjekk er Icarus(kapittel 3.5). Dette er et kraftig simuleringsverktøy, som også utfører en god sjekk av kodet. Siden vi ikke skal kjøre en simulering, men kun sjekke koden, benytter vi oss av kommandoen som er vist i kode 6.7.

KODE 6.7: Kode sjekk med Icarus

```
Iverilog fileName.v
```

Når vi kjører denne kommandoen returneres en beskrivelse av hva og hvor feilen i koden er, hvis det eksisterer en feil. Ved feil sender vi denne koden til frontend som presenterer feilmeldingen i en liten boks nede til høyre på skjermen.

Det må også legges inn eksempeldesign og kode-mal. For eksempeldesign velger vi å lage et eksempel hvor en lysdiode blinker. Eksempeldesignet som vi lager oss er vist i kode 6.8.

KODE 6.8: Eksempeldesign

```
module blink (
    // declaration of ports
        input  clk,     // all inputs
        output LED1     // all outputs
);
    // declaration of signals
        reg [31:0] counter;
    reg ledBlink;

    // initialization
    initial begin
    counter <= 32'b0;
    ledBlink <= 1'b0;
    end

    // register
        always @(posedge clk) begin
                counter <= counter + 1'b1;
        if (counter > 50000000) begin
                ledBlink <= !ledBlink;
                counter <= 32'b0;
        end
        end

    // output logic
```

```
        assign LED1 = ledBlink;
endmodule
```

Vi legger også inn en mal for hvordan Verilog-kode kan skrives. Dette er vist i kode 6.9

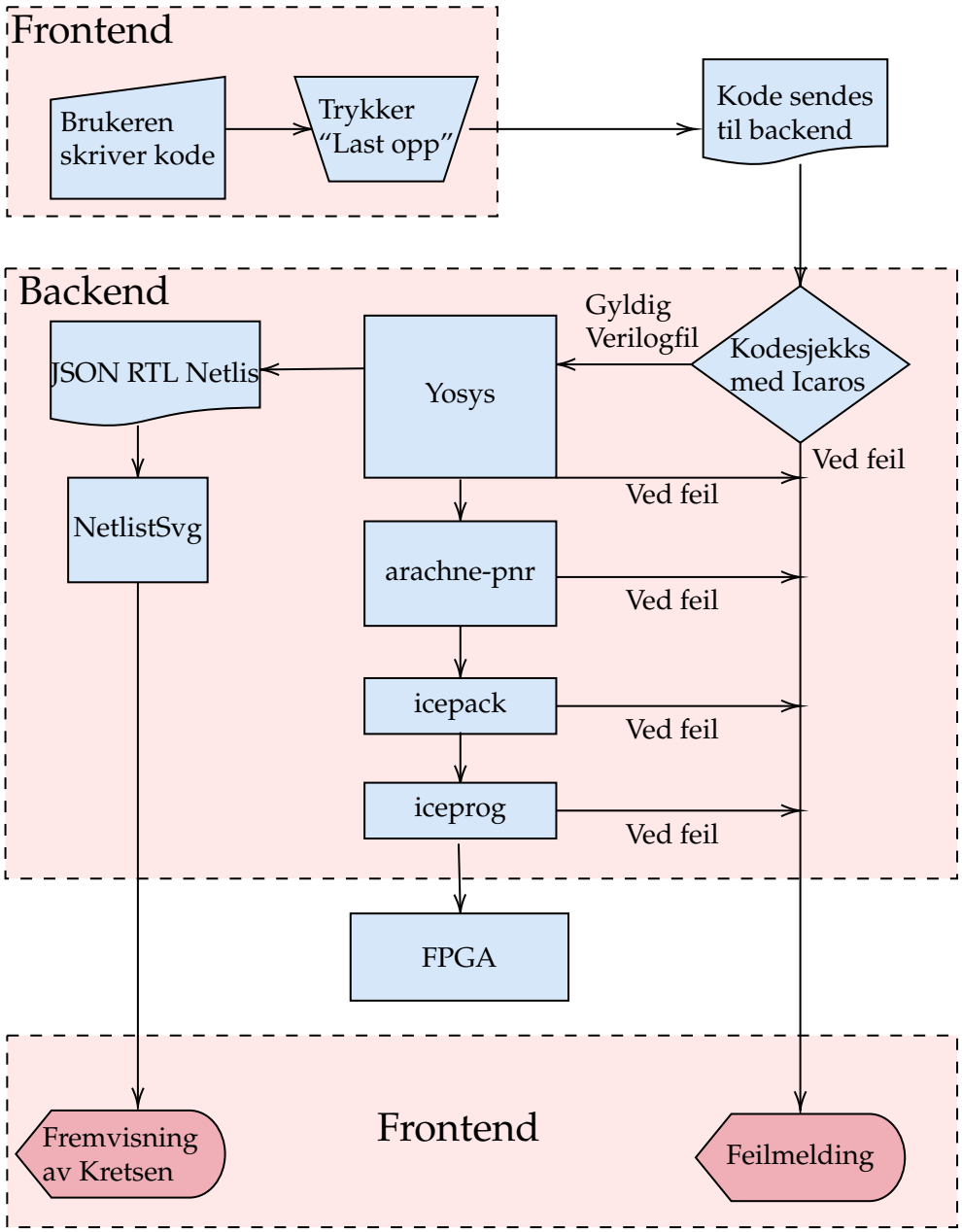
KODE 6.9: Kode-mal

```
module designName (
    // declaration of ports
    input clk, input2, input3,        // all inputs
    output out1, out2                 // all outputs
);

// declaration of constants
localparam yourConst = 2;

// declaration of signals
reg [2:0] signal;
reg signal2;

// initialization
initial begin
    signal <= 1'b0;
    signal2 <= 1'b0;
end

// register
always @(posedge clk) begin
    if (input2)
        signal <= input3;
    else
        signal2 <= 1'b1;
end

// output logic
assign out1 = signal;
assign out2 = signal2^yourConst;
```

```
endmodule
```

Med disse integreringene og forbedringene er den endelige prototypen av Dino ferdig. Hvordan flyten til programmet er avhenger av hvilken kommando som utføres. Den mest omfattende operasjonen skjer når man skal laste opp kode, da blir alle deler av programmet aktivert. Programflyten vil være som vist i figur 6.5 når brukeren trykker på "last opp" knappen.

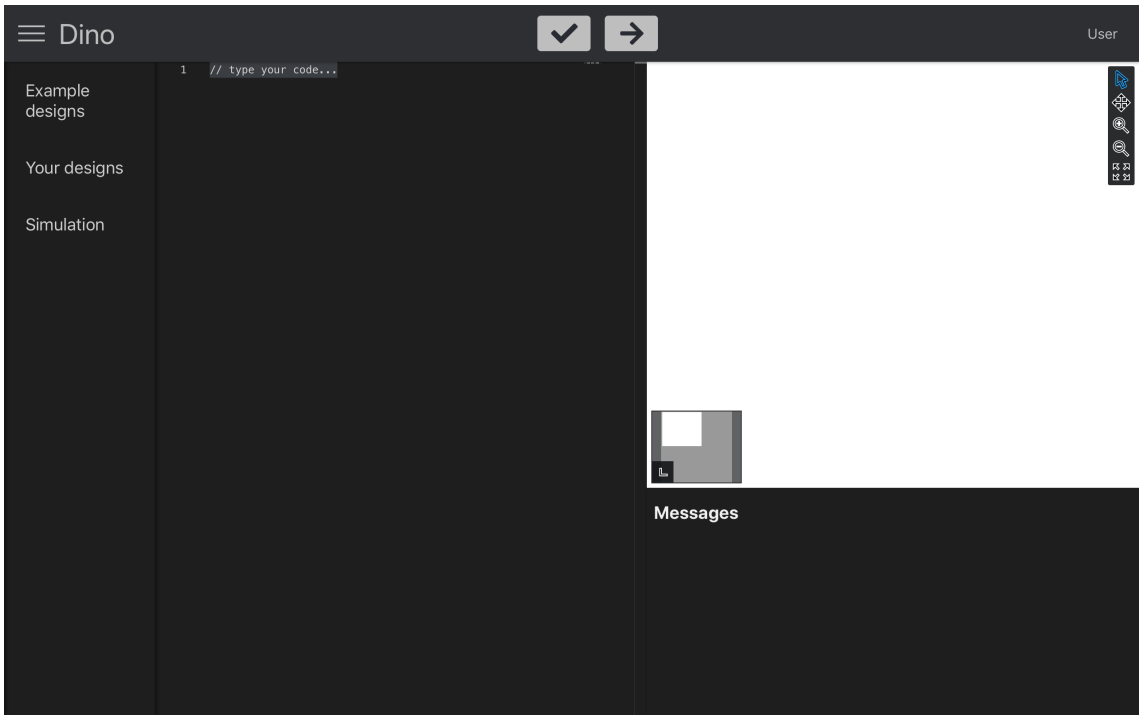
Filstrukturen til det endelige backendet er presentert i appendiks 8, og koden som er utviklet ligger i vedlagt zip-fil.



FIGUR 6.5: Programflyt for programmering av FPGA

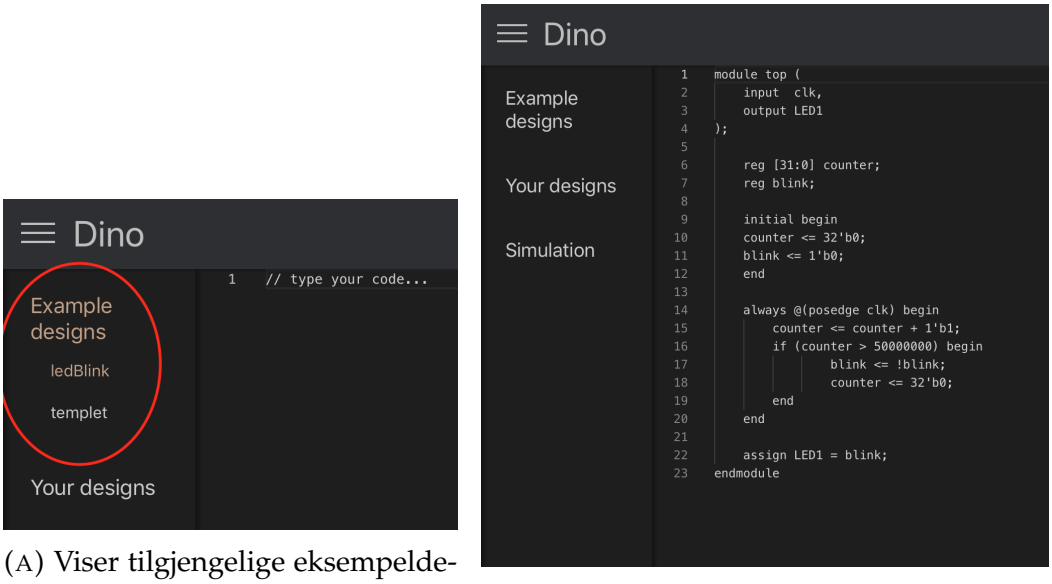
6.4 Evaluering

Nå som den endelige prototypen er ferdig, må vi evaluere den. For å kunne utføre en god evaluering, trenger vi å lage en plan for fremvisningen og hvordan vi ønsker at evalueringen skal gjennomføres. Vi starter på samme måte som for den første prototypen, å sette forventningene. Intervjuobjektene informeres om at dette er den endelige prototypen, og at det er et fungerende program. Verktøyet inneholder eksempeldesign, en mal for hvordan man organiserer en Verilog-fil, støtte for syntetisering og programmering av FPGA, og fremvisning av kretsdesign. Videre utfører vi en demonstrasjon av programmet. Demonstrasjonene er presentert under.



FIGUR 6.6: Startskjerm

Det første som møter brukeren er startskjermen som er vist i figur 6.6. Brukeren har nå flere valg, han/hun kan enten begynne å skrive Verilog kode i feltet hvor det står "type your code...", åpne ett av dine egne design i mappen "Your designs", eller du kan åpne et eksempel fra "Example design". Velger brukeren å åpne et eksempeldesign vil det se ut som i figur 6.7.

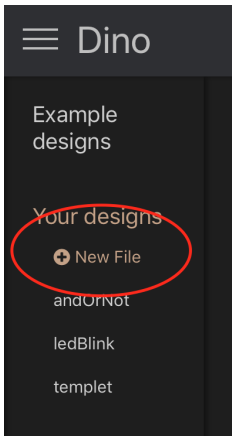


(A) Viser tilgjengelige eksempeldesign

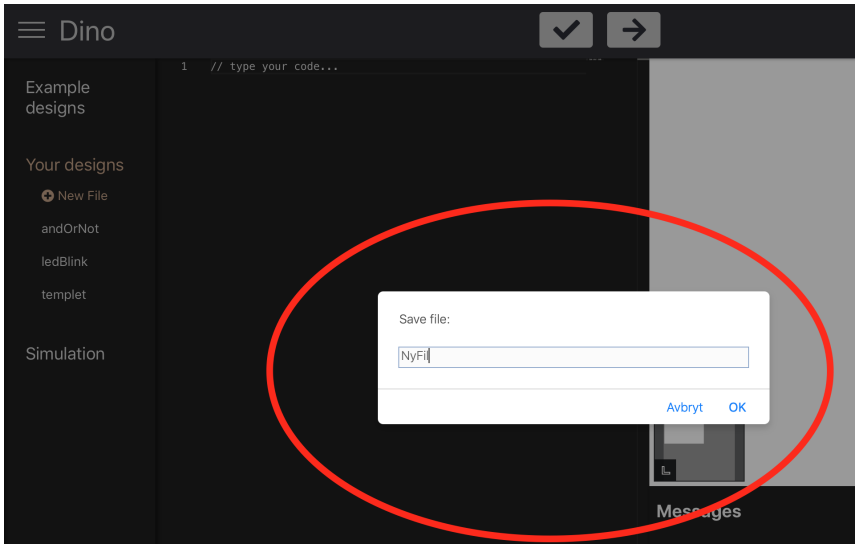
(B) Viser koden til ledBlink eksempel

FIGUR 6.7: Åpne eksempeldesign

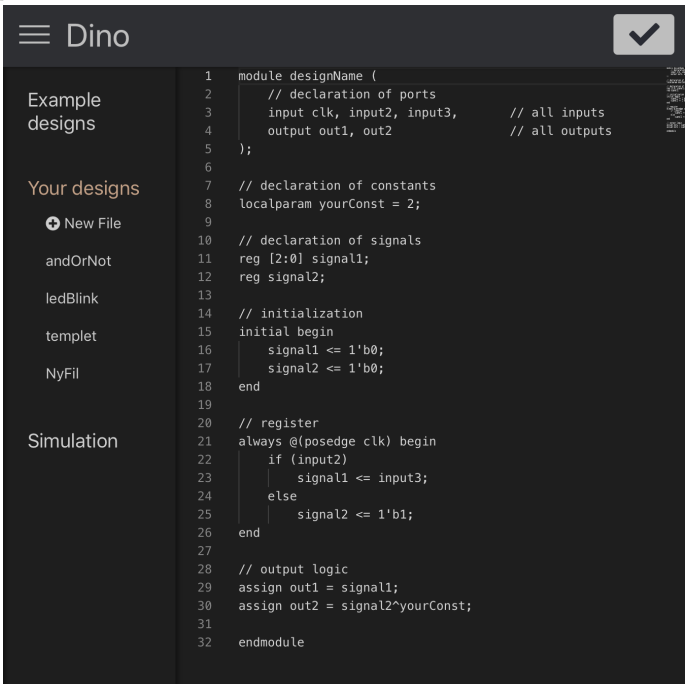
Ønsker brukeren å lage en ny fil trykker han/hun på "New File" under "Your designs"(figur 6.8a), da vil man få en melding om å gi filen et navn(figur 6.8b). Trykker man så ok, vil filen lagres og bli tilgjengelig på venstre side under "Your designs". Når man lager en ny fil blir også brukeren presentert med en mal på hvordan man kan strukturere en Verilog fil(figur 6.8c). Hele eksempelet er vist i figur 6.8



(A) Ny fil knapp



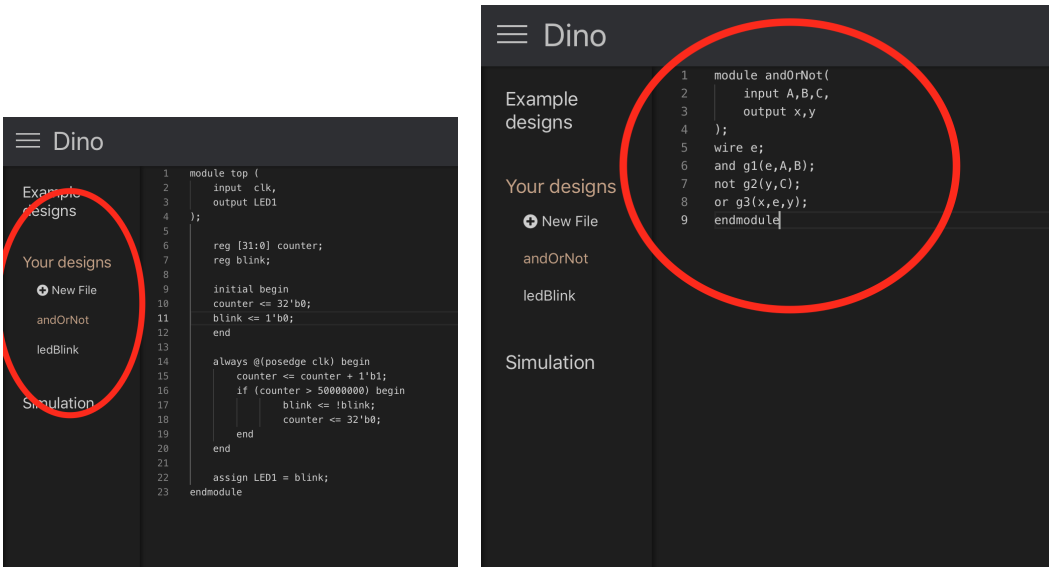
(B) Ber bruker om filnavn



(C) Viser ny fil med kode-mal

FIGUR 6.8: Lage en ny fil

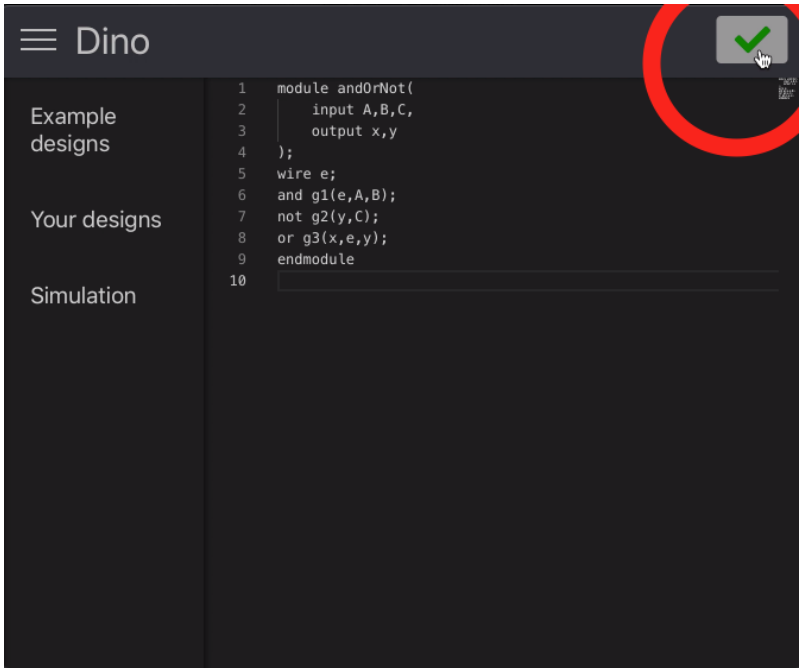
Ønsker brukeren heller å trykke på "Your designs" vil brukeren få fremvist alle tilgjengelige filer, og ved å trykke på ønsket fil vil denne bli tilgjengelig i tekstbehandlingsfeltet, som vist i figur 6.9



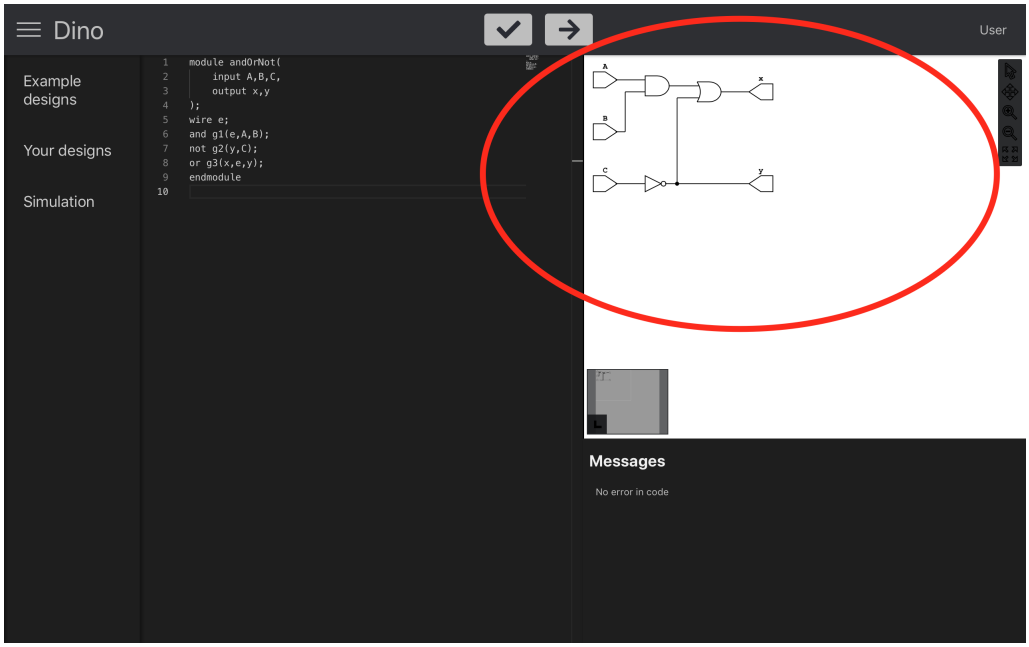
(A) Viser brukerens egne filer (B) Viser koden til andOrNot fil

FIGUR 6.9: Åpne eget design

Når brukeren er fornøyd med koden og ønsker å verifisere arbeidet trykker man på "verifiser" knappen som vist i figur 6.10a. Koden sendes så til backend for verifisering og hvis koden er gyldig vises kretsen som illustrert i figur 6.10b.



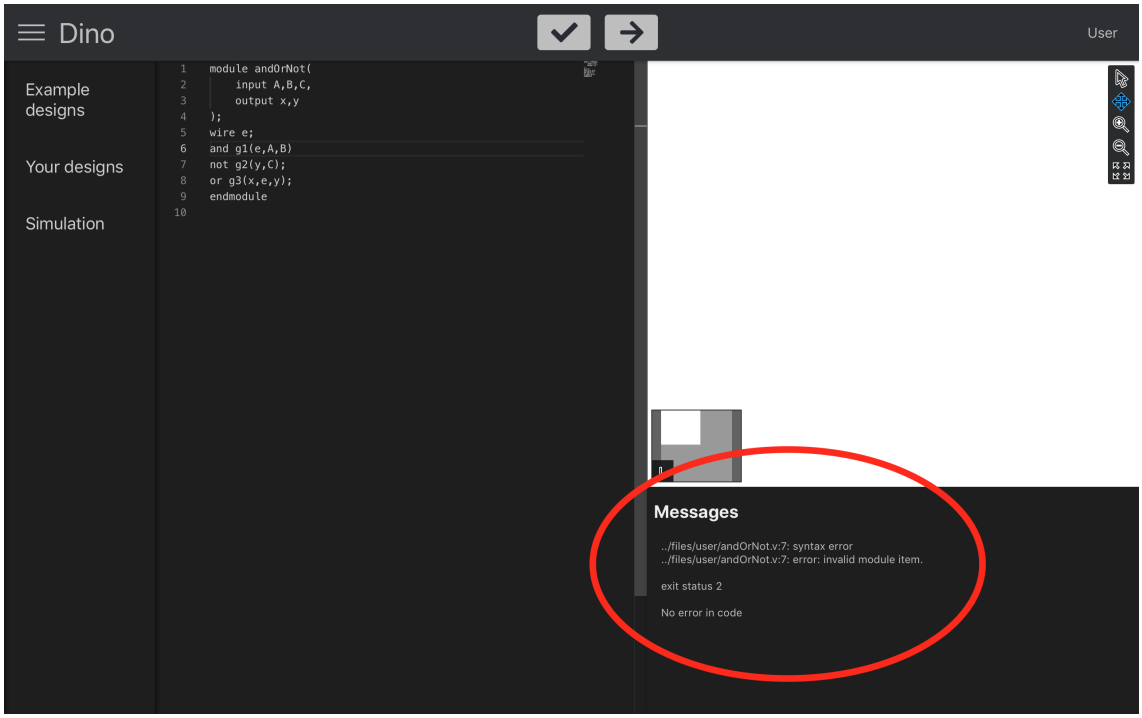
(A) Verifisere kode



(B) Fremvisning av kode

FIGUR 6.10: Verifisere og fremvisning av krets

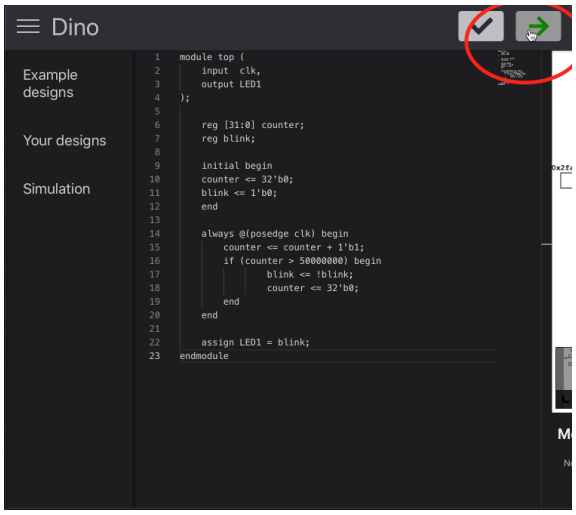
Hvis brukeren har feil i koden vil det komme opp i "Messages" feltet. Benytter vi koden i figur 6.10a og for eksempel fjerner en semikolon, vil koden bli ugyldig. Brukeren vil da få beskjed om at det er en syntaksfeil og i hvilken linje denne feilen befinner seg. Dette er demonstrert i figur 6.11.



FIGUR 6.11: Demonstrasjon av syntaksfeil

Hvis brukeren ønsker å laste koden opp på en FPGA, må han/hun først koble utviklingskortet med ønsket FPGA inn i PC-en, som vist i figur 6.12a. Videre må brukeren velge kode og trykke på knappen med pil(figur 6.12b). Hvis brukeren velger å kjøre eksempeldesign ledBlink, så vil lysdioden på utviklingskortet begynne å blinke(et lite rødt lys på kortet, figur 6.12c). Denne demonstrasjonen er illustrert i figur 6.12.

(A) Koble inn kort



(B) Trykke på "last opp" knappen

(C) ledBlink kodeeksempel kjører på kortet

FIGUR 6.12: Demonstrasjon: Laste opp kode til FPGA

Etter at demonstrasjonene er gjennomført og intervjuobjektene får testet ut programmet selv, blir de stilt noen spørsmål om opplevelsen av programmet. Følgende spørsmål blir stilt:

- Hva synes du var bra med programmet?
- Hva synes du var svakhetene og hva bør bli forbedret?
- Er programmet enkelt å bruke, og hvorfor?
- Tror du en person med lite kunnskap om digital design vil kunne bruke det?
- Kan man lære noe av å bruke dette programmet?
- Ser du et potensial i programmet?

6.5 Resultat

På det førte spørsmålet om hva intervjuobjektene likte beste ved programmet, var det enkelte egenskaper som gjentok seg. Det som utmerket seg som den beste egenskapen til programmet var enkelheten. Alle intervjuobjektene var enige om at programmet var svært brukervennlig og intuitivt. Programmet var strukturert på en oversiktlig måte, og med kun to knappetrykk kan man overføre kode til en FPGA. Intervjuobjektene som hadde minst erfaring med maskinvareprogrammering S1 og S2 trakk frem eksempeldesign og kodemalen som et svært godt tilskudd, som gjorde det lettere å starte sitt eget design. Student S3 og professor P1 var veldig begeistret for den raske responsen i programmet og at fremvisningen av kretsdesignet kom på kort tid.

Etter at styrkene ved programmet ble diskutert, ble intervjuobjektene spurt om svakhetene. Flere av intervjuobjektene trakk frem mangelen av fargekode som en svakhet, da dette vil øke lesbarheten av koden. Intervjuobjektet S1 foreslo at en visualisering av hvordan signaler sendes gjennom kretsen hadde vært til god hjelp. S2 og S3 ønsket litt mer funksjonalitet når det kom til tekstbehandlingsfeltet. S2 ønsket muligheten til å åpne flere faner med ulik kode i, og S3 ønsket muligheten for å søke etter enkelt ord i alle eksisterende filer. S3 mente også at programmet ville blitt styrket med en mer avansert feilmeldingsdeteksjon. Det hadde vært en fordel hvis man fikk et skille mellom store feil og advarsler. P1

mente at verktøyet fungerte veldig bra og hadde ingen ting å utsette.

På spørsmål om programmet var enkelt å bruke, var alle enige om at det stemte. At det var få knapper og valgmuligheter gjorde opplevelsen veldig intuitiv, og det var enkelt å velge et eksempeldesign å laste den opp på FPGAen. Professoren P1 trakk frem Verilog-malen som en av styrkene når det kom til at programmet var enkelt å bruke.

Det var stor enighet om at dette programmet kunne brukes av personer med liten kunnskap om digital design. På spørsmålet om man kan lære noe av å bruke programmet var alle enige. S1 som selv ikke har erfaring med digital design bekreftet at verktøyet hjalp på forståelsen. Bare ved å åpne eksempeldesign og fjerne og legge til linjer med kode og deretter se utsalg på den grafiske fremvisningen var til god hjelp. P1 tror den raske overgangen fra kode til fungerende design på et fysisk utviklingskort vil hjelpe på motivasjonen og nysgjerrigheten for ufaglærte personer. P1 hadde ikke sett en tilsvarende rask løsning før.

Potensialet til verktøyet var alle enige om at var til stede. S3 mente det var godt egnet til små prosjekter som DIY, og S1 tror det kan være til stor nytte for personer som ikke går elektronikk studier. S2 så potensialet i verktøyet, men at det var avhengig av om FPGA blir mer almen kjent. Professor P1 var svært begeistret, og mente at programmet hadde stort potensialet. Professoren så for seg at dette verktøyet var noe som professoren selv gjerne kunne benyttet i undervisningssammenheng.

Alle intervjuobjektene mente at verktøyet kunne bli benyttet av dem selv, men S3 ville mest sannsynlig valgt et program som allerede var kjent for personen, og som lagrer filer lokalt, da lokal fillagring var viktig for S3. S2 kunne brukt verktøyet til hobbyprosjekter.

Kapittel 7

Diskusjon

Når man lager en utviklingsplattform som Dino, er det en rekke valg som blir gjort underveis. Disse valgene kan gjøre produktet enten bedre eller dårligere. Dette kapittelet tar derfor for seg styrkene og svakenhetene ved Dino og diskuterer valgene som har blitt gjort.

7.1 Frontend

7.1.1 Brukergrensesnitt

Det første som ble utviklet i denne oppgaven var den visuelle delen. Vi benyttet resultatene fra Dino prosjektoppgaven(kapittel 3.8) som utgangspunkt. Fra denne oppgaven fikk vi vite hva slags elementer som skulle være med, blant annet et tekstbehandlingsfelt og en visualisering av kretsdesign. Fra den oppgaven fikk vi også vite at det var et ønske om enkelt og gjenkjennelig grensesnitt. Vi valgte derfor å implementere Monaco editor(samme som Visual Studio Code), og benyttet kun to knapper for å utføre verifisering og opplasting av kode. Dette var inspirert fra det enkle grensesnittet som Arduino har. Fra intervjuene som ble gjennomført i denne oppgaven virket det som at denne løsningen fungerte bra. Alle var svært positive til organiseringen av grensesnittet og at programmet var intuitivt å bruke. Allikevel er det noen endringer som kunne blitt gjort. Monaco er et godt tekstbehandlingsverktøy, men når det kommer til Verilog-kode finnes det ingen språklig støtte. Dette fører til at vi ikke får en fargeindikasjon på koden vi skriver samt ingen autokomplettér, sånn som det finnes hos andre tekstbehandlingsprogrammer. Monaco har funksjonalitet for å legge til egne språk. Det betyr at det hadde vært mulig å lage egen fargeindikasjon og autokomplettér for Verilog-kode, men dette var det dessverre ikke tid til. Dette kunne også ha løst seg hvis man hadde benyttet et annet tekstbehandlingsverktøy som Theia, da de

allerede har støtte for maskinvarespråk som Verilog. På den andre siden var professoren P1 fornøyd med at det ikke var så mye farger, og mente at farger kunne gjør det forstyrrende.

7.1.2 Fremvisning av krets

Når det kommer til fremvisning av krets, fungerer den nåværende løsningen fint på enkle og små design. For enkle design gir det en god hjelp med å forstå hvordan komponenter er koblet sammen. Blir designene større, oppstår det utfordringer. Slik Dino er nå, blir store kretser veldig uoversiktlige. Derfor burde det vært mulighet til å kunne endre hierarkiet til den visuelle kretsen. Slik at man kan bestemme om man vil se moduler eller de logiske elementene. Hvis dette var implementert hadde det også vært en stor fordel med en mer interaktiv kretsfremvisning. Brukeren kunne ved å trykke på den ønskede modulen forflytte seg inn og ut av hierarkiet, og få et lavnivå eller høynivå representasjon av designet. Denne funksjonaliteten var det ikke tid til å implementere. I tillegg var det heller ingen støtte for dette i Netlistsvg. Netlistsvg har allikevel støtte for at man kan legge til egne tegninger av kretselementer, så det hadde vært mulig å lage sine egne elementer, og den nødvendige logikken slik at denne interaksjonen kunne blitt en realitet.

7.2 Backend

7.2.1 Verktøy for FPGA utvikling

Når det kommer til verktøyene som skal overføre Verilog-kode til en kjørbarkode på FPGAen finnes det flere muligheter. Løsningene som ble vurdert i denne oppgaven er VisualHDL(kapittel 3.1), Concept Engineering(kapittel 3.2), VTR kapittel 3.6) og Prosjekt IceStorm(kapittel 3.3). Alle disse løsningene er svært gode, men de har sine fordeler og ulemper. VisualHDL er et integrert utviklingsmiljø som allerede gjør flere av de løsningen som skulle utvikles i denne oppgaven. Dette er i utgangspunktet bra, men ulempen er at dette også ville begrenset mulighetene for å lage et godt grensesnitt. En annen svakhet med denne løsningen var at man benytter et C++ lignende språk, og Dino var tenkt at skulle benytte et maskinbeskrivende språk som Verilog eller VHDL. Neste mulighet var Concept Engineering, som er et velutviklet verktøy for visualisering av digitale design. Verktøyet de har laget innehar mange løsninger som Dino gjerne kunne hatt.

Problemet er at verktøyet ikke har åpen kildekode, noe som gjør at man ikke får mulighet til å integrere dette inn i Dino. VTR er neste løsning. Dette er et komplett verktøy for å konvertere Verilog til ruting. VTR ser ut som et godt verktøy, som kunne blitt benyttet i denne oppgaven. Løsningen ble ikke implementert på grunn av mangelen for grafisk fremstilling, da dette er en viktig del av Dino. Den løsningen som stilte med flest funksjonaliteter var Prosject IceStorm. Her får man synteseverktøy, plassere og ruting, programmering av brikke og mulighet for fremvisning av krets. Dino bruker Yosys fra Prosjekt IceStorm som syntese-verktøy. Et annet synteseverktøy som ble vurdert var Odin II(kapittel 3.6) som er en del av VTR. Dette verktøyet utfører syntese og konverterer Verilog kode til en blif fil, og kunne erstattet Yosys når det kommer til å syntetisere og overføre kode til Arachne-pnr. Allikevel hadde det vært et behov for Yosys. Ser vi på figur 6.5 som viser programflyten, har Yosys to oppgaver. I tillegg til å utføre syntese trenger vi å konvertere designet til JSON-objekt, slik at Netlistsvg kan generere en grafisk representasjon. Av denne grunn må vi uansett benytte oss av Yosys, og det vil være overflødig å implementere Odin II, da Yosys allerede må benyttes.

7.2.2 Simulering

Som man ser i det endelige designet, figur 6.6, er det et felt på venstre side hvor brukeren kan lage simuleringsfiler. Som beskrevet kapittel 6 var simulering lavt prioritert, siden simuleringsfunksjonalitet ble vurdert som et hjelpemiddel som ville være positivt å ha, men ikke avgjørende for å realisere Dino. Fokuset under utviklingen av Dino var å lage et oversiktlig grensesnitt, hvor korrelasjonene mellom maskinvarespråk og grafisk fremvisning skulle øke brukerens forståelse av kretsens oppførsel. I retrospekt kan det se ut som den grafiske fremstillingen ikke bidrar like mye til læring som først antatt, siden kretsene fort blir uoversiktlige. Derfor kan det se ut som at simulering ville vært en god funksjonalitet for å øke forståelsen av hvordan tid spiller inn på kretsen, og for å verifisere designets oppførsel.

7.2.3 Utviklingskort

Dino har til nå støtte for ett utviklingskort, iCEstick med iCE40 FPGA. For å demonstrere funksjonaliteten og potensialet til Dino holder det med støtte for et kort. Med dette kortet kan man lage design med både inn- og utdata og lysdioder. Skal verktøyet nå et større publikum og skal man lage mer omfattende design, vil

det være nødvendig å legge til støtte for flere utviklingskort. For å legge til flere kort er man avhengig av at disse kortene er støttet av syntetisering, plassere og rute, og programmerings verktøyene. Prosjekt IceStorm har støtte for flere utviklingskort, men det må være kort med en iCE40 FPGA. Dette betyr at Dino vil kun ha støtte for denne FPGAen hvis man skal benytte Prosjekt IceStorm.

7.3 Eksempelkode

Eksempelkode er et godt hjelpemiddel når man skal lære seg noe nytt. Fra en undersøkelse som så på effekten av bruk av eksempler når man skulle lære seg ulike program api, ble det funnet ut at eksempelkode reduserer feil, øker suksessraten og utvikleren får en bedre opplevelse [67]. Den samme opplevelsen hadde også flere av testpersonene under demonstrasjonen av Dino-prototypen. Denne prototypen stiller kun med et eksempeldesign, noe som kan være litt lite, men på den andre siden har brukeren også tilgang på en mal som hjelper med kodeoppsett. Det skal også nevens at flere kodeeksempler er tilgjengelig under "Your designs", men disse er ikke eksempler som utnytter lysdioder eller inndata på FPGAen, og blir derfor ikke ansett som fullverdige eksempler.

7.4 Utviklingsprosess

Utviklingen av Dino ble gjennomført i to prototyputviklinger, den første prototypen tok for seg brukergrensesnittet og den andre prototypen knyttet brukergrensesnittet sammen med nødvendig maskinvareverktøy til en fungerende interaktiv prototype. Disse to prototyputviklingene ble gjennomført på en grundig måte, hvor alle valg underveis ble vurdert og begrunnet. Med tanke på utviklingstiden, var to prototyprunder det mest hensiktsmessige, da hver runde tar mye tid. Allikevel ville man med flere runder fått en grundigere evaluering av Dino, og man kunne hatt mulighet til å implementere enda flere funksjonaliteter som ville økt brukeropplevelsen.

Kapittel 8

Konklusjon og videre arbeid

Gjennom denne oppgaven har målet vært å skape en utviklingsplattform kalt Dino for FPGA design. Plattformen Dino skulle være rettet mot personer med liten erfaring innen digital design og FPGA utvikling. Det skulle derfor være et stort fokus på brukergrensesnittet. Det skulle være enkelt og intuitivt siden eksisterende programmer ofte er avanserte og rettet mot industrien. Under utviklingen av Dino skulle man vurdere hvordan designrepresentasjonen skal være, hva slags funksjonalitet det skal inneha, og bruken av eksempeldesign. For å realisere dette på en mest mulig hensiktsmessig måte, har teori rundt programvareutvikling blitt flittig brukt. Gjennom denne utviklingen ble det først laget en modell av ønsket design. Modellen var inspirert av Arduino, og fokuset var brukervennlighet, få visuelle elementer, og logisk programstruktur. Denne modellen ble evaluert gjennom intervju og spørsmål. Fra evalueringen kom det tydelig frem at designet på programmet var enkelt og intuitiv. Modellen hadde lyktes med å være minimalistisk, og intervjuobjektene likte godt den grafiske fremvisningen av koden. Dette var en god løsning som ville hjulpet brukeren med å forstå relasjon mellom kode og digital krets.

Siden resultatene fra første prototype var positive, var neste steg å lage en kodet interaktiv prototype. Denne prototypen skulle realisere designet fra første prototype og legge til all nødvendig funksjonalitet slik at brukeropplevelsen ble størst mulig. Prototypen ble utviklet ved hjelp av eksisterende designverktøy for FPGA utvikling. For å kunne overføre maskinvarebeskrivende kode til en FPGA ble prosjekt IceStorm benyttet. Med prosjekt IceStorm fulgte det med syntetiseringsverktøyet Yosys, plassere og rute-verktøyet Arachne-PNR, og et toolkitt for overføring av bitstrøm til FPGA. Verktøyene som ble brukt fungerte tilstrekkelig for å demonstrere potensialet til Dino, men for å øke brukeropplevelsen ytterligere trengs det mer avanserte løsninger. Den første svakheten med denne løsningen

er deteksjon av feil i kode. Yosys har ingen støtte for dette, og det ble derfor benyttet et ekstra verktøy kalt Icarus som håndterte kode sjekk. Verktøyet fungerte godt til å finne syntaksfeil, men klarte ikke å skille mellom store feil og mindre syntaksfeil. En mer avansert løsning kunne hjulpet brukeren med å forstå hva som er bra og dårlig kode, ikke bare hva som er feil. Dino hadde også en god grafisk fremvisning av krets som ble realisert gjennom et verktøy kalt Netlistsvg. Denne løsningen fungerte fint for små design, men ved større design ble kretsen fort uoversiktlig. Av den grunn ville det vært en fordel å ha implementert en løsning med mulighet for å forflytte seg i et hierarki. Ved at den visuelle kretsen først representeres i moduler, som brukeren kan trykke på for så å se innholdet i denne modulen.
For å øke brukeropplevelsen enda mer, ble det også implementert kodeeksempel og kode-mal som brukeren kunne bruke som utgangspunkt. Fra evalueringen som ble gjort kom det tydelig frem at dette var et godt tilskudd, og noe som var til stor nytte hvis man mangler den generelle kunnskapen om digitaldesign.

I alt ble det utviklet en velfungerende prototype som demonstrerte potensialet med et slikt utviklingsverktøy. Verktøyet var godt egnet for ufaglærte personer og med alle hjelpemidlene som fremvisning av krets, kodeeksempel og deteksjon av kodefeil lyktes programmet med være pedagogisk og lærerikt.

Programmet har stort potensialet, og markedet for et slikt verktøy er til stedet. Det er flere ting som bør jobbes videre med for å forbedre Dino. Først bør tekstebehandlingsfeltet gjøres bedre, funksjonalitet som fargemarkering av kode, samt autokomplettér vil øke brukeropplevelsen. I tillegg kan det være en fordel å implementere deteksjon av syntaks feil inn i tekstebehandlingsfeltet også. Den grafiske fremvisningen av kretsen kan også jobbes videre med. Her ville det vært en stor fordel med en mer interaktiv løsning. Dette ville gjort opplevelsen av denne funksjonen mer oversiktlig og intuitiv. For å hjelpe ufaglærte brukere vil det også være en fordel å implementere enda flere eksmepler som gjør det lett å se bruskområder og hvordan man skriver kode.

Bibliografi

[1] L. E. Skraastad, «Dino», Desember 2018.

[2] Arduino, *arduino - introduction*. side: https://www.arduino.cc/en/Guide/Introduction#.

[3] E. Monmasson og M. N. Cirstea, «FPGA Design Methodology for Industrial Control Systems—A Review», *IEEE Transactions on Industrial Electronics*, årg. 54, nr. 4, s. 1824–1842, aug. 2007, ISSN: 0278-0046. DOI: 10.1109/TIE.2007.898281.

[4] A. Khandale og H. Bhagyalakshmi, «Low Power FPGA Architecture», 2013.

[5] T. Qin, S. Bleiker, S. Rana, F. Niklaus og D. Pamunuwa, «Performance Analysis of Nanoelectromechanical Relay-Based Field-Programmable Gate Arrays», *IEEE Access*, årg. PP, s. 1–1, mar. 2018. DOI: 10.1109/ACCESS.2018.2816781.

[6] N. Instruments. (nov. 2018). FPGA Fundamentals, side: http://www.ni.com/white-paper/55015/en/.

[7] J. Rose og V. Betz, «How Much Logic Should Go in an FPGA Logic Block?», *IEEE Design & Test of Computers*, årg. 15, s. 10–15, jan. 1998, ISSN: 0740-7475. DOI: 10.1109/54.655177.

[8] F. Piltan, O. Avatefipour, S. Soltani, O. Mahmoudi, M. Reza, S. Nasrabad, M. Eram, Z. Esmaeili, S. Heidari, K. Heidari og M. Mahidi Ebrahimi, «Design FPGA-Based CL-Minimum Control Unit», *International Journal of Hybrid Information Technology*, årg. 9, s. 101–118, jan. 2016. DOI: 10.14257/ijhit.2016.9.1.10.

[9] C. Wolf. (). Yosys Manual, side: http://www.clifford.at/yosys/files/yosys_manual.pdf (sjekket 14.05.2019).

[10] J. Bhasker, *Verilog HDL synthesis: a practical primer*. Star Galaxy Publishing, 1998, ISBN: 0965039153.

[11] D. Jansen, *The electronic design automation handbook*. Springer Science & Business Media, 2010, ISBN: 1402075022.

[12] D. C. Black, J. Donovan, B. Bunton og A. Keist, *SystemC: From the ground up*. Springer Science & Business Media, 2009, bd. 71.

[13] A. Prost-Boucle, O. Muller og F. Rousseau, «A fast and autonomous hls methodology for hardware accelerator generation under resource constraints», i *2013 Euromicro Conference on Digital System Design*, IEEE, 2013, s. 201–208.

[14] Y. Explorations, «eXCite C to RTL Behavioral Synthesis 4.1 (a)», *Y Explorations (YXI)*, 2010.

[15] T. Feist, «Vivado design suite», *White Paper*, årg. 5, s. 30, 2012.

[16] D. A. S. Committee mfl., «IEEE Standard Verilog (R) Hardware Description Language [Z]», *IEEE Computer Society, http://standards. ieee. org*, 2001.

[17] ——, *IEEE Computer Society, IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS Changes), Std. 1076.1*, 1997.

[18] L. Bening og H. Foster, *Principles of verifiable RTL design*. Springer, 2001, ISBN: 9780306476310.

[19] N. A. Sherwani, *Algorithms for VLSI physical design automation*. Springer Science & Business Media, 2012, ISBN: 0792383931.

[20] *Digital design and computer fundamentals*, eng, Harlow, 2013.

[21] V. Udar og S. Sharma, «Analysis of place and route algorithm for field programmable gate array (FPGA)», i *2013 IEEE Conference on Information Communication Technologies*, apr. 2013, s. 116–119. DOI: 10.1109/CICT.2013.6558073.

[22] B. N. Arnowitz Jonathan Arent Michael, *Effective Prototyping for Software Makers*, ser. Interactive Technologies. Elsevier Science, 2006, ISBN: 0120885689.

[23] E. R. Ivar Liseter, «HTML», nov. 2018. side: https://snl.no/HTML.

[24] E. Rossen, «CSS», nov. 2009. side: https://snl.no/CSSL.

[25] D. Flanagan, *JavaScript: the definitive guide*. "O'Reilly Media, Inc.", 2006.

[26] I. Liseter, «server», nov. 2018. side: https://snl.no/server.

[27] H. Dvergsdal, «HTTP», nov. 2012. side: https://snl.no/HTTP.

[28] I. Fette og A. Melnikov, «The websocket protocol», tekn. rapp., 2011.

[29] D. Mellis, M. Banzi, D. Cuartielles og T. Igoe, «Arduino: An open electronic prototyping platform», i *Proc. Chi*, bd. 2007, 2007. side: https://alumni.media.mit.edu/~mellis/arduino-chi2007-mellis-banzi-cuartielles-igoe.pdf.

[30] Arduino. (). Products, side: https://www.arduino.cc/en/Main/Products (sjekket 29.05.2019).

[31] M. Banzi og M. Shiloh, *Getting started with Arduino: the open source electronics prototyping platform*. Maker Media, Inc., 2014, ISBN: 9781449363338.

[32] (). Which Arduino Is Best for Your Project, side: https://www.digikey.com/en/maker/blogs/2018/which-arduino-is-best-for-your-project (sjekket 29.05.2019).

[33] Arduino. (). Compare board specs, side: https://www.arduino.cc/en/Products/Compare (sjekket 29.05.2019).

[34] ——, (). Download the Arduino IDE, side: https://www.arduino.cc/en/Main/Software (sjekket 18.12.2018).

[35] ——, (). Serial, side: https://www.arduino.cc/reference/en/language/functions/communication/serial/ (sjekket 18.12.2018).

[36] ——, (). Getting Started with Arduino Web Editor on Various Platforms, side: https://create.arduino.cc/projecthub/Arduino_Genuino/getting-started-with-arduino-web-editor-on-various-platforms-4b3e4a?f=1 (sjekket 18.12.2018).

[37] A. A. Galadima, «Arduino as a learning tool», i *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)*, sep. 2014, s. 1–4. DOI: 10.1109/ICECCO.2014.6997577.

[38] Arduino. (). Serial, side: https://www.arduino.cc/reference/en/ (sjekket 18.12.2018).

[39] Sysprogs. (). VisualHDL, side: http://sysprogs.com/legacy/visualhdl/ (sjekket 31.05.2019).

[40] ——, (). About THDL++, side: http://sysprogs.com/legacy/visualhdl/thdlpp/ (sjekket 31.05.2019).

[41] (2018). We make things visible, side: http://www.concept.de/index.html (sjekket 31.01.2019).

[42] (2018). RTLvision PRO, side: http://www.concept.de/RTLvision.html (sjekket 31.05.2019).

[43] C. Wolf og M. Lasser, *Project IceStorm*, http://www.clifford.at/icestorm/.

[44] C. Wolf, *Yosys Open SYnthesis Suite*, http://www.clifford.at/yosys/.

[45] C. Wolf, J. Glaser og J. Kepler, «Yosys-a free Verilog synthesis suite», i *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, 2013.

[46] U. Berkeley, «Berkeley logic interchange format (BLIF)», *Oct Tools Distribution*, årg. 2, s. 197–247, 1992.

[47] C. Wolf. (). The Verilog front-end does not create nice errors for invalid code, side: http://www.clifford.at/yosys/faq.html (sjekket 14.05.2019).

[48] C. Seed. (2018). Arachne-pnr, side: https://github.com/YosysHQ/arachne-pnr (sjekket 28.05.2019).

[49] N. Turley. (). netlistsvg, side: https://github.com/nturley/netlistsvg (sjekket 31.05.2019).

[50] S. Williams. (). Introduction, side: https://iverilog.fandom.com/wiki/Introduction (sjekket 31.05.2019).

[51] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose og V. Betz, «VTR 7.0: Next Generation Architecture and CAD System for FPGAs», *ACM Trans. Reconfigurable Technol. Syst.*, årg. 7, nr. 2, 6:1–6:30, jun. 2014.

[52] P. Jamieson, K. B. Kent, F. Gharibian og L. Shannon, «Odin II - An Open-Source Verilog HDL Synthesis Tool for CAD Research», i *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, mai 2010, s. 149–156. DOI: 10.1109/FCCM.2010.31.

[53] A. Mishchenko. (). ABC, side: https://people.eecs.berkeley.edu/~alanmi/abc/ (sjekket 11.06.2019).

[54] V. Betz og J. Rose, *VPR: a new packing, placement and routing tool for FPGA research*, W. Luk, P. Y. K. Cheung og M. Glesner, red., Berlin, Heidelberg, 1997.

[55] (). Trådløse Trondheim, side: https://wirelesstrondheim.no (sjekket 20.04.2019).

[56] A. Singh, S. Sharma, S. R. Kumar og S. A. Yadav, «Overview of PaaS and SaaS and its application in cloud computing», i *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*, feb. 2016, s. 172–176. DOI: 10.1109/ICICCS.2016.7542322.

[57] K. Aalde, «FPGA Development in The Cloud Using The IDE8 Developer Framework», 2018.

[58] V. Studio. (). Visual Studio, side: https://code.visualstudio.com.

[59] M. Granevang. (2016). Frontend, side: https://snl.no/Frontend (sjekket 08.05.2019).

[60] ——, (2015). Backend, side: https://snl.no/Backend (sjekket 08.05.2019).

[61] D. W. C. Finch, R. Ruelas, A. G. Potes og R. A. Santos, «REACT: An Object-Oriented Control Engine for Rapidly Building Control Systems», i *Electronics, Robotics and Automotive Mechanics Conference (CERMA 2007)*, sep. 2007, s. 123–128. DOI: 10.1109/CERMA.2007.4367672.

[62] Microsoft. (). Monaco-editor, side: https://microsoft.github.io/monaco-editor/ (sjekket 12.06.2019).

[63] Theia. (). Theia Cloud and Desktop IDE, side: https://www.theia-ide.org/doc/index.html (sjekket 11.06.2019).

[64] A. A. Donovan og B. W. Kernighan, *The Go programming language*. Addison-Wesley Professional, 2015, ISBN: 9780134190440.

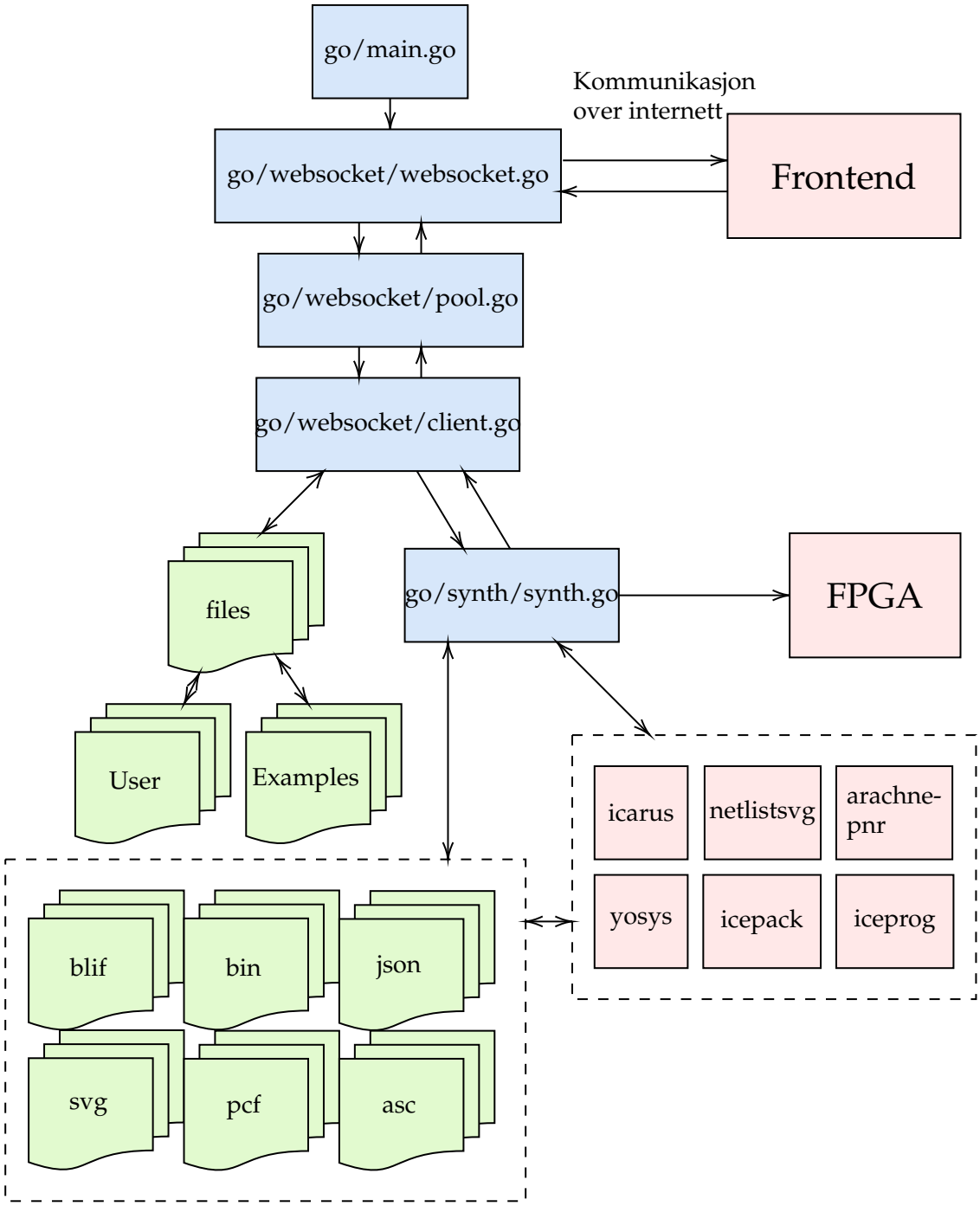
[65] Lattice. (). Icestick, side: https://www.latticesemi.com/icestick.

[66] C. Vadala. (). react-svg-pan-zoom, side: https://www.npmjs.com/package/react-svg-pan-zoom-nl.

[67] S. M. Sohan, F. Maurer, C. Anslow og M. P. Robillard, «A study of the effectiveness of usage examples in REST API documentation», i *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, okt. 2017, s. 53–61. DOI: 10.1109/VLHCC.2017.8103450.

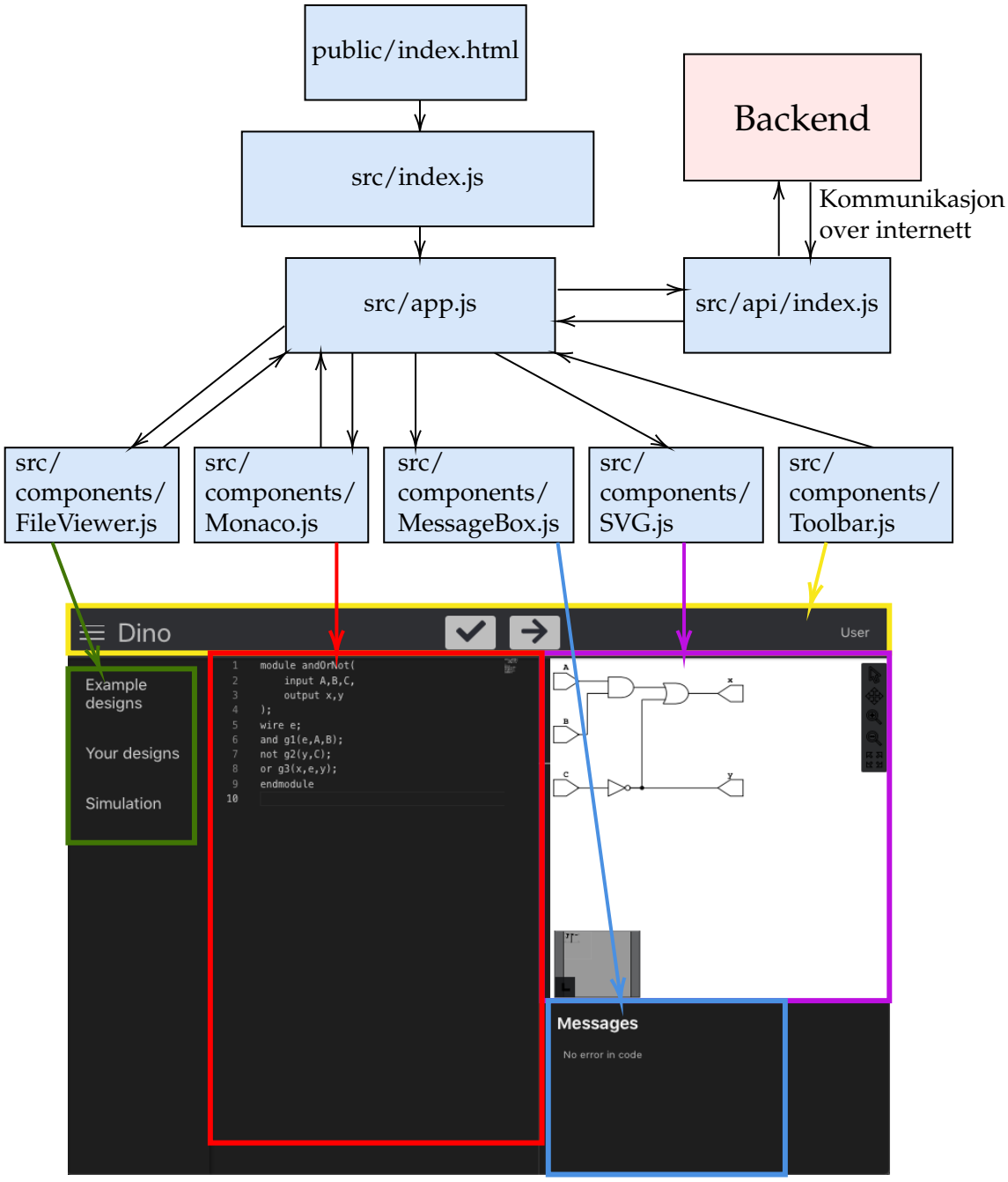
Tillegg A

Filstrukturen til backend



Tillegg B

Filstrukturen til frontend



Lars Erik Tiset Skraastad

Uino – Plattform for forenklet FPGA-design

NTNU
Norwegian University of
Science and Technology